Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Zrimšek

Izdelava fizikalnega pogona za računalniške igre

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Divjak Saša
Somentor: doc. dr. Matija Marolt

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Zrimšek, z vpisno številko **63100279**, sem avtor diplomskega dela z naslovom:

Izdelava fizikalnega pogona za računalniške igre

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Divjaka Saše in somentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 14. septembra 2014 Podpis avtorja:

Zahvalil bi se mentorju doc. dr. Divjaku Saši in somentorju doc. dr. Matiji Maroltu za pomoč in usmerjanje pri izdelavi diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
FPS	frames per second	število sličic na sekundo
2D	two dimensional	dvodimenzionalno
3D	three dimensional	trodimenzionalno
BB	bounding box	mejna škatla
SAT	separating axis test	test ločevanja osi

Povzetek

Diplomsko delo predstavi fiziko in matematiko, potrebno, za izvedbo fizikalnega pogona, primernega za uporabo v realnočasovnih aplikacijah. Pri razvoju smo se osredotočili na računalniške igre in zato upoštevali omejitve in pasti, ki se lahko pri tem pojavijo. Ob spoznavanju različnih področij fizike bomo povedali tudi, na kaj moramo paziti pri vključitvi teh v fizikalni pogon in kako jih realizirati. Na začetku predstavimo namen in uporabo fizikalnih pogonov in pa omenimo že obstoječe rešitve. Nato začnemo z lažjimi temami, ki jih kasneje nadgrajujemo. Najprej spoznamo delce brez mase, ki nam omogočajo uporabo balistike, sile v splošnem in pa vzmeti, kot posebno vrsto le-teh. Nato se spoprimemo s trdnostnimi omejitvami, kot so detekcije trkov, razreševanje trkov v primeru kolizij teles in pa njihovo gibanje v takšnih situacijah. Tako dobimo masovno celotni fizikalni pogon, ki ga nadalje nadgradimo z rotacijami in tako ustvarimo pogon s togimi telesi. Tu razločimo pomembna koncepta sile navora in vztrajnostnega momenta, potrebna za samo realizacijo in pravilno obnašanje togih teles. Za konec predstavimo način implementacije fizikalnega pogona, uporabo pogona pa demonstriramo z aplikacijo, ki vsebuje štiri scene, ki predstavijo različne podmnožice v tem diplomskem delu razloženih fizikalnih vidikov.

Ključne besede: računalniške igre, fizikalni pogon, simulacija, detekcija trkov, manipulacija objektov, 3D, realnočasovna aplikacija.

Abstract

The thesis presents the physics and mathematics necessary for the realization of the physics engine, suitable for use in real-time applications. In the development process we have focused on computer games and therefore have taken into account the limitations and pitfalls that can arise in such a case. Upon learning about different fields of physics, we will also make a notice on what caution must be taken when integrating these into a physics engine and how they are realized. We start by presenting the purpose and use of physics engines and also mention already existing solutions. Then we tackle with the minor themes, which can later be upgraded. First of all, we get to know with particles without mass, which enable us to realize ballistics, forces in general and springs, as a special type of them. Then we confront the hard constraints such as collision detection, resolving collisions in the case of collisions of bodies and their movements in such situations. This gives us a mass aggregate physics engine, which can be further upgraded with rotations and thus create a physical engine with rigid bodies. Here we distinguish important concepts of force and moment of inertia, required for the realization and the correct behaviour of rigid bodies. At the end, we explain the implementation details of the constructed physics engine and demonstrate its usage with an application, that contains four scenes that present different subsets of physical aspects, explained in this thesis.

Keywords: computer games, physics engine, simulation, collision detection, object manipulation, 3D, real-time application.

Poglavje 1

Uvod

Na področju potrošniške računalniške grafike se je v zadnjih letih zelo povečala uporaba fizikalnih pogonov, ki omogočajo realno obnašanje navideznega sveta v realnem času. V današnjih igrah so postali že standard, ki dajo igri dodano vrednost in tako igralcem še dodatno zameglijo mejo med realnim in navideznimi svetovi.

Posnemanje realne fizike je že popolnoma samoumevno v današnjih računalniških igrah. Predstavlja zelo pomemben dejavnik v načinu igranja, saj nas oddalji od vnaprej posnetih animacij, kjer je izid vedno enak in nam dopušča večjo svobodo pri interakciji z okolico in drugimi objekti, s čimer se lahko oddaljimo od togih iger in vključimo dinamiko, ki omogoča ustrezno obnašanje igre glede na našo interakcijo.

V okviru diplomske naloge se bomo dotaknili ozadja izvedbe raznih področij fizikalnih simulatorjev v področju računalniških iger in jih implementirali, izbrana pa bomo tudi demonstrirali v obliki aplikacije oz. računalniške igre. Pri sami realizaciji simulatorja je zelo pomemben dejavnik zahtevnost izračunov, potrebnih za simulacijo, torej se bomo morali poslužiti algoritmično učinkovitih metod za realizacijo samega pogona, ki bodo karseda ekonomično izkoriščale procesorsko moč, vendar hkrati omogočale zadovoljive rezultate.

1.1 Pregled področja

Sicer je sama uporaba fizike prisotna v igrah že od nekdaj. Sprva so se je posluževali predvsem pri simulaciji dinamičnih sistemov delcev za dim, eksplozije, vode in podobnih pojavov, nato pa so jo začeli uporabljati tudi pri statičnih objektih, predvsem v povezavi z mehaniko, s čimer oživimo dinamiko celotne igre, kar so sprva koristili predvsem pri raznih simulatorjih za avtomobilsko, letalsko ali drugo vožnjo. Vendar pa je v zadnjem času doživela pravi razcvet in nas skupaj z vedno bolj grafično bogatimi in realističnimi slikami iger pripeljala še korak bliže navideznemu svetu, ki je vedno bolj podoben pravemu. Vse to je možno predvsem na račun vedno zmogljivejših računalnikov, z močnejšimi procesorji, še zlasti tistimi, ki so prisotni na sami grafični kartici, namenjenimi skoraj izključno računanju operacij časovno zahtevnih grafičnih aplikacij, kar fizikalni pogoni še kako s pridom izkoriščajo. Zelo pomemben aspekt pa so postale tudi simulacije mehkih objektov, kot so oblačila in dlake, od nedavnega pa tudi model lutke, s katerim posnemamo okostje objekta, kar pride zelo koristno pri interaktivni animaciji posameznih udov trupel.

Povpraševanje po tovrstnih izdelkih se je pred nekaj leti začelo hitro povečevati in na trgu so se pojavili zelo napredni fizikalni simulatorji. Fizika predstavlja ogromno podkategorij, med drugim detekcijo trkov, kinetiko za gibanje objektov, optiko za simuliranje svetlobe, itd., v igrah pa se največ uporablja mehanika, kjer se upoštevajo razne sile, kot so na primer težnost, vztrajnost, plovnost in prožnost, pri samem gibanju in deformaciji objektov.

Na trgu obstaja veliko zmogljivih in priročnih rešitev. Velika podjetja po navadi razvijejo svoj lastni pogon za igre *game engine*, čeprav jih danes vse več posega po rešitvah, ki so na voljo na trgu, saj je trud, ki ga je potrebno vložiti v razvoj tovrstnih aplikacij, precejšen. Med najbolj znane predstavnike zagotovo spadajo *Unreal*, *Unity* in *Frostbite*. Pri manjših podjetjih ali samostojnih razvijalcih iger *indie game developers* pa je uporaba obstoječih aplikacij za razvoj iger bolj pravilo kot izjema. Skoraj povsem brez izjem pa se

tudi velika podjetja, ki se ukvarjajo z razvijanjem iger, odločijo za uporabo že obstoječih rešitev za simulacijo fizikalnih zakonov znotraj igre. Ti so namreč še mnogo bolj kompleksni kot sami pogoni iger. Najbolj znani odprtokodni fizikalni pogon, najbolj priljubljen v zadnjem času, je zagotovo *Havok*. Med popularnimi pa lahko zasledimo tudi *Bullet*, ali pa *Box2D*, ki pa omogoča le 2D fizikalne simulacije. Vsi pravkar omenjeni so odprtokodni in zato privlačni tudi za samostojne razvijalce. Med plačljivimi pa je najbolj cenjen *PhysX* iz podjetja *Nvidia*.

Ker je področij fizike nešteto, seveda ne moremo predstaviti vseh. Med zanimive spadajo recimo še deformiranje teles [8] in/ali gibanje le-teh [16], koncepti pri realizaciji resničnih obnašanj tkanin in oblek [6], pojavi prisotni pri mehaniki [15] in dinamiki [3] ter s tem povezani problemi [13], pa še bi lahko naštevali.

Simulacija fizikalnih pojavov je lahko koristna tudi na drugih področjih, razen v igrah. Taka področja so na primer gradbeništvo [5], vojska [14] in zdravstvo oziroma znanost [11].

Za pregled zgodovine uporabe simulacij v igrah glej [2]. Kot referenco za hiter pregled pomembnih formul in njihov kontekst uporabe, ki so relevantne za področje iger, bomo uporabili članek [9].

1.2 Motivacija

Sama tema je danes zelo aktualna, ne samo na področju računalniških iger, ampak tudi pri raznoraznih simulacijskih programih, namenjenim področjem vseh vrst, od zabavne industrije, treniranja vojakov ter voznikov letal, ladij in avtomobilov, do simulacije fizikalnih vplivov na stavbe v arhitekturi ali na kakovost komponent in delovanje strojev v strojništvu. Spoznanja analize tovrstnega področja bomo uporabili pri implementaciji lastnega sistema za simuliranje fizikalnih lastnosti, ki se bo lahko uporabljalo pri realno časovnih grafičnih aplikacijah, torej tudi v igrah.

Cilj tega diplomskega dela je ustvariti fizikalni pogon v obliki, katerega funkcionalnosti bomo demonstrirali z izgradnjo demonstracijske igre, ki ga bo vključevala.

1.3 Potek dela

Na začetku se bomo posvetili obravnavi najbolj prakticiranih pristopov k implementaciji posameznih vidikov fizikalnega simulatorja, ki jih bomo uporabili tudi sami pri lastni implementaciji. Nato bo sledila teorija iz področij matematike, fizike in programiranja, ki nas bo pripravila na implementacijo pogona. Osredotočili se bomo predvsem na področje trkov med objekti različnih preprostejših oblik, kjer bomo za dosego karseda realističnega obnašanja upoštevali nastavljive parametre fizikalnih sil posameznih teles, vključenih v simulacijo.

Sama gradnja fizikalnega pogona bo sestavljena iz več tematsko zaokroženih celot, ki bodo predstavile teorijo, samo prakso pa bomo predstavili z demonstracijsko aplikacijo oz. igro. Najprej si bomo pogledali nekaj osnovnih zadev, ki jih bomo po korakih nadgrajevali in dopolnjevali. Kot prvo bomo izvedli fiziko delcev, ki bo skrbela za gibanje oz. transformacijo točk in jo kasneje razširili v fiziko t. i. masovnih celot, ki jih ti delci sestavljajo. Temu bomo dodali rotacije in rotacijske sile in tako bomo ustvarili pogon z *popolnoma togimi telesi*. Nato se bomo posvetili kontaktom in pa detekciji trkov med objekti. Na koncu pa bomo še obravnavali problem razreševanja trkov oz. dotikov med objekti.

Skozi proces izdelave diplomske naloge, predvsem dela povezanega s samo implementacijo fizikalnega pogona in z njo povezano grafično aplikacijo, se bomo opirali na naslednji knjigi [4] in [12]. Kot priporočilo za hiter pregled pomembnih formul in njihov kontekst uporabe, ki so relevantne za področje iger, bomo uporabili članek [9]. Pri sami teoriji in orisu problema, s katerimi se srečamo, pa bomo preučili članke, ki se nanašajo na različna izbrana področja fizikalnih simulacij in tako pridobili poglobljeno specifično znanje le-teh.

1.4 Tipi fizikalnih pogonov

Obstaja več različnih pristopov izdelave fizikalnih pogonov, kjer ima vsak svoje slabosti in prednost. Ena od lastnosti ki si jo moramo izbrati je tip objektov, ki bodo nastopali v pogonu.

Ločimo pogone s t. i. *popolnoma togimi telesi (full rigid bodies)* in *celostno masnimi (mass aggregate)* pogoni. Prvi obravnavajo oz. simulirajo telesa oz. objekte kot celoto (*kot zaboj (crate)*) in se smatrajo za nedeformljive (geometrično nespremenljive), pri drugem tipu pa so objekti sestavljeni iz več manjših mas (masnih točk oz. delcev), ki kot mreža obdajajo celotno telo in so med seboj povezane z (neupogljivimi) drogovi (rod). Naj poudarimo, da lahko z vpeljavo logike delcev in pa sil celostno masnemu pogonu omogočimo tudi prožne povezave med delci. Prav tako lahko pri togih telesih s primerno logiko spreminjamo njihovo obliko z relativnim premeščanjem delcev, ki sestavljajo togo telo, pri čemer ignoriramo njegovo gibanje kot celoto. Celostno masni pogoni ne potrebujejo rotacij, saj je vsaka masna točka pravzaprav čisto navadna točka v prostoru, zato so lahko izračuni za transformiranje izraženi z linearnimi enačbami, celoten objekt pa se primerno transformira zaradi povezav med samimi masami oz. točkami. Ampak to povzroča, da telesa niso popolnoma toga in postanejo nekoliko prožna, zato je potrebno kar precej dodatne kode, da to pomanjkljivost odpravimo. Ker pa je lažji za implementacijo, se bom lotil implementacije le-tega, in mu nato dodali rotacije in ga tako razširil v pogon s popolnoma togimi telesi.

Odločiti se moramo tudi, kako bomo obravnavali stike med vsemi objekti (nekateri so lahko med seboj povezani, spet drugi se dotikajo oz. počivajo na nekem drugem objektu itd.). V tej diplomski nalogi se bomo poslužili ti. *iterativnega pristopa*, kjer se pregleda telesavsakega objekta z vsakim. Ta pristop je hiter, vendar pa je dejstvo, da lahko kontakti vplivajo drug na drugega. Ti. *Jakobinsko (Jacobian)*[1] *orientiran pristop* vzame tudi to v upoštev, saj upošteva

[1]Način za matematično predstavitev posledic enega kontakta na drugega.

vse vplive na objekte kot celoto, vendar pa je časovno zelo kompleksen. Tu so lahko računi tako kompleksni, da včasih ni pravilnega rezultata, tako da potrebujemo dodatno kodo za tovrstne primere. So najbolj pogosti v komercialnem svetu. Najbolj realističen pa je pristop z *reduciranimi koordinatami*, kjer uporabimo nabor enačb, s katerimi določimo zakone fizike za posamezne objekte, ki pa se spreminjajo za vsako novo izračunano sliko (frame). Uporablja se predvsem v inženirstvu, za realnočasovne aplikacije pa zaradi izjemne počasnosti ni primeren.

Ko se objekti med seboj dotaknejo, moramo kontakte nekako tudi razrešiti. *Silno orientirani* pogoni uporabljajo sile in delujejo podobno kot v resničnem svetu; *sunkovno orientirani* pa sunke sil[2]. Uporabili bomo drugi pristop, ker so manj časovno potratni. Za vse nepremične predmete vsako sliko opravi majhne trke, da jih lahko ohrani na mestu, za razliko od konstantnih sil pri prvem pristopu. Edina težava se pojavi pri zmanjšanju hitrosti *sličc na sekundo (FPS - frames per second)*, kjer lahko nepremični predmeti začnejo vibrirati. Ampak v večini primerov se dobro odreže in je skoraj ekvivalenten silno orientiranemu pogonu.

[2] `http://sl.wikipedia.org/wiki/Sunek_sile`

Poglavje 2

Delci

2.1 Fizikalni zakoni delcev

Ker delamo fizikalni pogon, bomo za začetek upoštevali Newtonove zakone gibanja, saj želimo obnašanje v našem navideznem svetu karseda podobno pravemu.

Točkovna masa, ali kakor jo v svetu iger imenujejo *delec (particle)*[1], ima maso, ne pa tudi velikost, kar pomeni, da ji lahko pripišemo pozicijo, rotacijo pa prav tako ne. Tega v naravi ne zasledimo in tako so delci sami po sebi nerealni in neuporabni. Vendar pa jih lahko uporabimo za poenostavitev raznih fizikalnih konceptov. Delce bomo predstavili z vrsto lastnosti, med katerimi so najbolj osnovne pozicija, hitrost in njihov pospešek, ki pa so predstavljene kot vektorji. Njihovo obnašanje natančno opiše Newton s svojimi tremi zakoni o gibanju[2].

Za začetek se bomo osredotočili na prva dva. Če bi lahko nekemu predmetu odstranili vse sile, ki delujejo nanj, bi se ta gibal s konstantno hitrostjo v neskončnost. To pravi prvi zakon. Na Zemlji, predvsem zaradi sile *upora (drag)*,

[1] "Particles"imajo v svetu fizike popolnoma drug pomen in ne upoštevajo Newtonovih zakonov

[2] `http://csep10.phys.utk.edu/astr161/lect/history/newton3laws.html`

in seveda tudi povsod drugje, kjer je le-ta prisoten, temu ni tako in na predmet deluje nasprotna sila, ki ga upočasnjuje. To silo bi sicer lahko enostavno ignorirali in jo obravnavali posebej, vendar jo bomo zaradi zaokroževalnih napak računalnikov na tem mestu izkoristili za preprečenje, da ne bi predmeti pospeševali, namesto, ko bi morali pojenjati. Povedano drugače, v primeru, da nočemu telesu pripisati upora, mu bomo vseeno pripisali neko minimalno vrednost zanj z namenom rešitve numeričnih problemov.

Pred vsako izrisano sličico na zaslonu bomo z integriranjem izračunali nove položaje delcev, vendar pa jim bomo pred tem tudi proporcionalno zmanjšali hitrost glede na vrednost upora, ki deluje na njih. Parameter upornosti nam torej pove, kolikšen odstotek hitrosti ostane delcu od prvotne, prisotne pri prejšnjem simulacijskem koraku; vrednost 1 pomeni, da mu hitrost sploh ne pojenja, 0 pa da je predmet popolnoma pri miru. Torej vrednost 1 pomeni da na predmet ne deluje upor, vendar pa je zaradi prej omenjenih računskih problemov priporočljivo le-to nastaviti malce pod vrednost 1.

Drugi zakon pa nam pove kako sile spreminjajo gibanje telesa, natančneje njihov pospešek. Na podlagi tega zakona, lahko telesu spreminjamo pozicijo ali/in hitrost le indirektno, in sicer tako, da na telo vplivamo z neko silo in mu na ta način spreminjamo pospešek toliko časa, dokler ne dosežemo želene pozicije oz. hitrosti. S tem se efektivno znebimo sunkovitega/negladkega gibanja. Pri tem je ključni parameter masa telesa, kot je razvidno iz formule za silo, ki vključuje pospešek:

$$f = ma \tag{2.1}$$

Za naše potrebe bomo potrebovali izpeljani pospešek:

$$a = \frac{1}{m} f \tag{2.2}$$

Tudi sile lahko predstavimo z vektorji.

Delec mora imeti tudi lastnost, ki označuje njegovo maso. Telo z maso enako 0 bi povzročil deljenje z 0, kar pa ni možno, poleg tega pa bi v tem primeru imel objekt neskončen pospešek, kar je nerealno, tako da moramo to

preprečiti. Mase z neskončno vrednostjo pa lahko prikladno izkoristimo za nepremične objekte; na ta način "izničimo"[3] učinek pospeška in če ima objekt hitrost nastavljeno na 0, se le-tega ne bo dalo premakniti. Torej, ker hočemo računsko predstaviti neskončno, 0 pa ne, bomo zamenjali vlogi tako, da bomo vpeljali spremenljivko, ki bo predstavljala izraz $1/m$ - inverz mase. Tako imajo objekti z neskončno maso inverz mase enak 0, objekti z maso 0 pa bi imeli to vrednost enako neskončnosti, ki pa je tako ali tako v večini programskih jezikov ne moremo predstaviti. S tem tudi pridobimo na hitrosti, saj ne potrebujemo računati inverza.

Pomemben vidik fizikalnega pogona je tudi gravitacija. V realnem svetu ta deluje med vsemi objekti in je odvisna od njihovih mas in medsebojnih instanc. Ker pa na Zemlji prevladuje njena gravitacija v tolikšni meri, da drugih sploh ne občutimo, se lahko osredotočimo samo nanjo. V skladu s tem lahko primerno poenostavimo Newtonov zakon gravitacije:

$$f = G\frac{m_1 m_2}{r^2} \tag{2.3}$$

kjer sta m_1 in m_2 masi teles, med katerima računamo privlačnost oz. gravitacijo, r razdalja med njunima središčema, G pa *univerzalna gravitacijska konstanta*. Predpostavimo lahko da se masa Zemlje ne spreminja, prav tako ne spremenljivka r, saj je razdalja od površine Zemlje do njenega jedra tako ogromna, da na njeni površini skoraj ni razlike v gravitaciji med posameznimi višinskimi točkami, kar nam nanese naslednjo poenostavljeno formulo:

$$f = mg \tag{2.4}$$

kjer je m masa simuliranega objekta, g pa konstanta, ki predstavlja:

$$g = G\frac{m_{zemlje}}{r^2} \tag{2.5}$$

g je pravzaprav pospešek, tako da je formula enaka prej omenjeni formuli za silo, Enačba (2.1), pri omembi 2. Newtonovega zakona. Če Enačbo (2.4)

[3]Dejansko se lahko pospešek le zmanjša (oziroma v primeru neskončnosti do njega sploh ne pride)

vstavimo v Enačbo (2.2) dobimo:

$$a = g \tag{2.6}$$

kar pomeni, da bodo vsi objekti pospeševali enako hitro kot posledica gravitacije in masa pri tem ne bo imela vpliva (v resnici na Zemlji na telesa vpliva še upor, ki pa je neodvisen od gravitacije, ampak pri delcih ta nima bistvenega vpliva, tako da ga bomo zanemarili). Torej, kot smo ugotovili, vsaj za zdaj sploh ne potrebujemo parametra za sile, saj moramo predstaviti le silo težnosti, ki pa je pravzaprav pospešek.

Ta znaša približno $g = 10ms^{-2}$, vendar ga bomo malce povečali, kar je tudi v praksi v svetu iger, saj bi se nam sicer samo dogajanje v igri zdelo počasno. V večini igralnih pogonov je koordinatni sistem nastavljen tako, da y koordinata predstavlja višino, zato lahko gravitacijo predstavimo z naslednjim vektorjem:

$$\vec{g} = \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} \tag{2.7}$$

kjer je g znotraj vektorja prej omenjena vrednost gravitacije.

2.2 Integrator

Integrator v svetu iger predstavlja tisti del logike, ki skrbi za posodabljanje pozicije in hitrosti vsakega objekta v igri pred vsakim izrisom sličice (ali kaki drugi realnočasovni aplikaciji seveda). Pred tem pa mora ugotoviti pospešek vsakega objekta, od katerega je odvisna hitrost, pozicija pa je odvisna od obeh parametrov. Pomemben parameter je še časovni interval, ki je lahko fiksni, torej ga lahko poljubno nastavimo za posamezne situacije in ostane konstanten ali pa je variabilen in je enak času, ki preteče med izrisom dveh zaporednih sličic. Druga opcija je bolj zaželena, zaradi prenosljivosti med sistemi, vendar pa se lahko pojavi problem, če se časi med izrisi posameznih sličic precej

razlikujejo. Gibanje objektov postane sunkovito, vendar v takšnih primerih tudi brez tega pristopa igra tako ali tako ne bi delovala tekoče. Obstaja pa tudi opcija, kjer *izrisovalno zanko (drawing loop)* ločeno poganjamo z variabilnim intervalom posodabljanja (druga opcija), fizikalno logiko pa poganjamo v svoji *niti (thread)* s fiksnim intervalom posodabljanja (prva opcija), kar se ekstenzivno uporablja v medmrežnih igrah s fizikalnimi simulacijami. Mi bomo uporabili drugo opcijo.

2.2.1 Enačbi za posodobitev položaja in hitrosti

S pomočjo matematične operacije integriranja lahko na podlagi znane hitrosti in pretečenega časa izračunamo trenutni položaj objekta:

$$s = s + vt \tag{2.8}$$

Enako velja za izračun hitrosti ob znanem pospešku in pretečenem času:

$$v = v + at \tag{2.9}$$

Z izračunom integrala drugega reda nad pospeškom dobimo končno formulo za posodobitev pozicije objekta:

$$s = s + vt + a\frac{t^2}{2} \tag{2.10}$$

Ker pa je časovni interval med zaporednima sličicama zelo majhen, bo zadnji del enačbe, $a\frac{t^2}{2}$, posledično zanemarljivo vplival na spremembo pozicije, zato se ga bomo znebili, in uporabljali Enačbo (2.9).

Sedaj pa še formula za posodabljanje hitrosti. Pri Enačbi (2.8) moramo upoštevati še upor in tako dobimo:

$$v = vd + at \tag{2.11}$$

kjer d predstavlja upor. Ta oblika enačbe predstavlja težavo, namreč ob spremembi hitrosti izrisa sličic (FPS) bo na hitrost objekta upor posledično deloval različno - za tolikokrat, kolikokrat se bo povečalo/zmanjšalo število sličic

na sekundo, toliko večkrat/manjkrat bo upor zmanjšal hitrost objekta in bo objekt deležen različnega upora vsako izrisano sličico (če bo upor nastavljen na vrednost 1[4] pa seveda v nobenem primeru nebo vplival na hitrost), kajti vrednost upora je nespremenljiva in neodvisna od dolžine časovnega intervala, pretečenega med zaporednima sličicama. To rešimo z odvisnostjo upora od pretečenega časa na naslednji način:

$$v = vd^t + at \tag{2.12}$$

Torej, na ta način upor predstavlja odvzem hitrosti objektu vsako sekund in ne tako kot pri Enačbi (2.11) vsako izrisano sličico. V tej obliki pa zopet nastopi problem hitrosti računanja, kajti potenciranje racionalnih števil (števil v plavajoči vejici) z racionalnimi števili je relativno počasna računalniška operacija. Rešili bi jo lahko tako, da bi imeli vsi objekti v igri/pogonu enako vrednost upora in bi vrednost d^t rabili izračunati samo enkrat, uporabili pa bi jo nad vsemi objekti. S tem bi bili precej omejeni (vsi objekti z istim uporom), tako da bomo ostali pri Enačbi (2.12).

2.3 Balistika

Eno od številnih področji, kjer lahko uporabimo delce, je *balistika*. Na primer projektil, ki se izstreli iz nekega orožja, bo dejansko predstavljen kot prej definirani delec. Tu moramo zopet prilagoditi fizikalne lastnosti, ki vladajo v naravnem svetu. Hitrosti, s katerimi leti izstrelek iz orožij, so namreč tako zelo velike, da naboja igralcu sploh ne bi uspelo opaziti (če pa potrebujemo tudi izjemno hitre hitrosti gibanja delcev, je v takem primeru bolj primerno uporabiti premico v smeri izstrelka in preveriti, ali seka oz. naleti na kako oviro). Zato je treba tovrstne hitrosti kar nekajkrat pomanjšati, pri tem pa moramo upoštevati, kakšne mere uporabljamo v igri (npr. en kilometer v igri lahko predstavlja en meter v pravem svetu).

[4] V resnici malce pod 1 zaradi že omenjenih računskih problemov

Učinek, ki ga povzroči projektil, ko zadene oz. trči ob nekaj, je odvisen tako od njegove hitrosti kot mase, ti dve lastnosti pa sta med seboj neposredno povezani oz. odvisni preko enačbe:

$$e = mv^2 \tag{2.13}$$

kjer je e energija, m in v pa masa in hitrost delca. Torej, ker želimo zmanjšati hitrost delcev, moramo povečati njihovo maso sorazmerno enako glede na faktorsko spremembo v hitrosti, če želimo ohraniti njihov učinek oz. energijo, ki jo povzročijo v realnem svetu. Faktorsko spremembo mase izračunamo na sledeči način:

$$\Delta m = (\Delta v)^2 \tag{2.14}$$

kjer Δ pomeni spremembo med prvotno in novo vrednostjo hitrosti/mase kot faktor prvotne vrednosti. Za primer izračunajmo maso m' za naboj s prvotno maso $m = 10g$ in hitrostjo $v = 1.000\frac{m}{s}$, kateremu zmanjšamo hitrost na $v' = 20\frac{m}{s}$. Najprej potrebujemo faktorsko spremembao mase Δm:

$$\begin{aligned} \Delta m &= (\Delta v)^2 \\ \Delta m &= (\frac{v}{v'})^2 \\ \Delta m &= (\frac{1.000\frac{m}{s}}{20\frac{m}{s}})^2 \\ \Delta m &= 2.500 \end{aligned} \tag{2.15}$$

s katero pomnožimo prvotno maso:

$$\begin{aligned} m' &= \Delta m m \\ m' &= 2.500 * 10g \\ m' &= 25.000g \end{aligned} \tag{2.16}$$

Sedaj pa še preverimo, ali se energija po spremembi hitrosti dejansko ohrani:

$$\begin{aligned} e = mv^2 = 10g * (1.000\frac{m}{s})^2 = 10.000.000g(\frac{m}{s})^2 \\ e' = m'v'^2 = 25.000g * (20\frac{m}{s})^2 = 10.000.000g(\frac{m}{s})^2 \end{aligned} \tag{2.17}$$

Kot vidimo, obvelja $e' = e$.

Ker smo zmanjšali hitrost delcev, moramo ustrezno proporcionalno zmanjšati tudi gravitacijo, sicer bi bil njen vpliv na objekt večji za tolikokrat, kolikokrat se je zmanjšala njegova hitrost:

$$g_{objekt} = \frac{g_{Zemlje}}{\Delta v} \tag{2.18}$$

kjer je g_{Zemlje} gravitacija Zemlje (približno $10\frac{m}{s^2}$), g_{objekt} pa gravitacija določenega objekta v igri.

Poglavje 3

Sile

Poleg gravitacije si želimo v pogon vključiti seveda tudi poljubne druge sile, ki po možnosti delujejo sočasno nad nekim objektom. Po *D'Alembert-ovem principu za delce* je potrebno vse sile, ki delujejo na posamezen objekt le sešteti in ta seštevek nato uporabiti nad objektom[1]. Različnih tipov sil je ogromno, od takih, kateresami upravljamo (dodajanje plina pri avtomobilu), takih, ki delujejo na objekte eksterno (kot je na primer sila gravitacije), do takšnih, ki so posledica drugih sil (premikanje objekta ustvarja v kombinaciji z gravitacijo upor in trenje (ob stiku s podlago)), zato je smiselno implementirati sistem, ki nam omogoča hitro in priročno dodajanje novih sil in jih efektivno prilagajati našim potrebam.

Ena od zapletenejših sil za implementacijo je med drugim upor. Pravi izračun je mnogo preobremenjujoč za realnočasovno procesiranje, tako da bomo uporabili poenostavljeno obliko enačbe za upor, ki upošteva številne domneve:

$$f_{upor} = -\hat{v}(k_1|v| + k_2|v|^2) \tag{3.1}$$

pri čemer sta k_1 in k_2 *koeficienta upora*, ki določata samo moč upora oz. njegov tip, v je hitrost telesa, $|v|$ je dolžina vektorja v in pomeni magnitudo hitrosti, $\hat{v}$ pa predstavlja normalizirano hitrost v, torej usmerjenost hitrosti. Z ustre-

[1]To seštevanje se opravi za vsako sličico posebej, pred samim integriranjem

zno nastavitvijo obeh koeficientov lahko predstavimo poljuben tip upora oz. vsaj njegov približek. Poleg tega je, kot vidimo v gornji enačbi, ta sila odvisna tudi od hitrosti telesa (v resnici pa še od številnih drugih parametrov), nad katerim deluje. Upor deluje v nasprotni smeri premikanja objekta (razvidno iz enotskega vektorja hitrosti $\hat{v}$, ki razkrije smer apliciranja upora nad telesom). Sama količina premikajočemu telesu nasprotno delujoče sile pa je odvisna od obeh omenjenih koeficientov, ki določata obnašanje upora, in se veča z večanjem hitrosti telesa. Koeficient k_2 ima poseben pomen in je uporaben pri vključitvi aerodinamike - za neničelni koeficient k_2 velja, da bo pri večji hitrosti telesa upor hitreje, eksponentno naraščal (pri skorajšnjem mirovanju na telo deluje zanemarljivo velik upor, pri večjih hitrostih pa lahko ogromno prispeva k njegovemu zaviranju).

Ker je eden naših primarnih ciljev izdelati hiter pogon, bomo pri gravitaciji in uporu obdržali prejšnji pristop, kjer smo ju direktno aplicirali na objekt, saj je vsakršno računanje pri tako splošnih silah, katerih obnašanje poznamo vnaprej, povsem odveč in predstavlja nepotrebno dodatno koriščanje procesorske moči. V primeru, da želimo spremenjeno različico teh dveh sil, ju še vedno lahko naknadno ustvarimo preko ustreznega vmesnika.

Poglavje 4

Vzmeti

Vzmeti so le ena od sil, ki se za svoje delovanje lahko poslužijo delcev. Njihova logika je prisotna pri vseh telesih, ki se upirajo deformaciji. Z vključitvijo le-teh se nam odpre mnogo večji nabor apliciranja fizikalnega pogona, od simulacije vode, oblek, mehkih in razgradljivih objektov ipd.

4.1 Izračun

Pri uporabi vzmeti bomo uporabljali slednjo formulo:

$$f = -k\Delta l \tag{4.1}$$

kjer je k *konstanta vzmeti*, ki določa togost oz. okornost vzmeti, Δl pa nam pove spremembo v dolžini raztegnjenosti/skrčenosti vzmeti glede na njeno prvotno stanje in dejansko pomeni $l - l_0$:

$$f = -k(l - l_0) \tag{4.2}$$

pri čemer je l_0 dolžina vzmeti v mirujočem stanju, l pa trenutna dolžina vzmeti. To je *Hook-ov zakon*, ki pravi, da je sila odvisna izključno od proporcionalne stisnjenosti oz. raztegnjenosti vzmeti glede na njen mirujoči položaj[1]. Ta sila

[1]Ali drugače, če bomo vzmet raztegnili za njeno dvakratno dolžino v prvotnem stanju, bo ta proizvedla silo dvakrat večjo tisti, ki nastopa ob prvotnem stanju

velja za oba konca vzmeti (sila potiska iz obeh strani vzmeti proti njenemu središču v primeru njene raztegnjenosti in izven središča v smeri obeh koncev v primeru njene skrčenosti).

Ker hočemo vključiti tudi take vzmeti, ki ne delujejo le enodimenzionalno, torej, so lahko poljubno orientirane znotraj prostora, moramo vzeti v upoštev kar celoten vektor namesto skalarja, ki zajema vse dimenzije našega (3D) prostora:

$$f = -k(|\vec{d}| - l_0)\vec{d} \tag{4.3}$$

kjer je $\vec{d}$ vektor, ki se razteza med obema koncema vzmeti, $|\vec{d}|$ pa njegova dolžina. Ta vektor je usmerjen proti tistemu koncu vzmeti, nad katerim produciramo silo vzmeti, torej:

$$\vec{d} = x_A - x_B \tag{4.4}$$

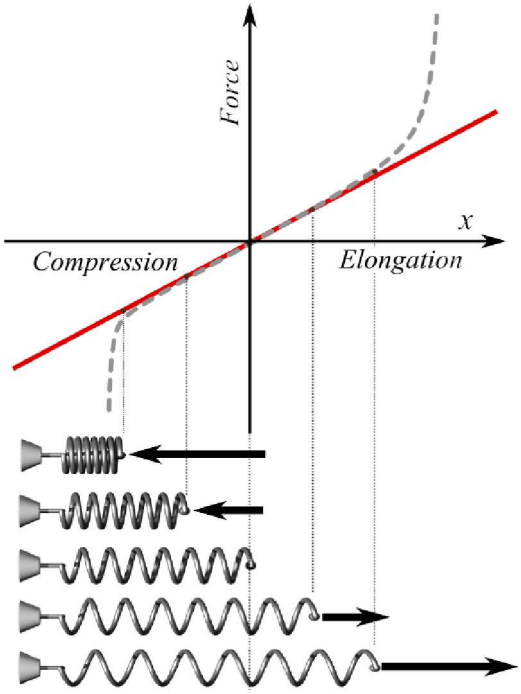
pri čemer je x_A predstavlja položaj tistega konca vzmeti, ki je pritrjen na objekt nad katerim deluje sila vzmeti, x_B pa ustreza položaju nasprotnega konca. Enačbo (4.3) moramo uporabiti za oba konca vzmeti, lahko se pa poslužimo dejstva, da je sila na nasprotnem koncu vzmeti enaka nasprotni sili prvega konca, saj imata oba konca enakomerno uravnoteženi sili, ki delujeta v nasprotujočih si smereh.

V resnici za vzmeti velja Enačba (4.2) le v omejenem intervalu dolžin, ki ji pravimo *limita elastičnosti* in se razlikuje med različnimi vzmetmi (odvisno od njenega materiala, oblike itd.). Če vzmet raztegnemo preko njene limite elastičnosti, se le-ta deformira, pri krčenju pa se ustavi na točki, ko je povsem stisnjena vase in njeno obnašanje postane enako togim telesom. Obnašanje v takih primerih je zelo kompleksno in zato nepraktično za vključitev v računalniške igre.

Vzmeti lahko razčlenimo po njihovem obnašanju: nekatere proizvedejo le poteg (*bungee* elastika), druge tudi potisk (vijačna vzmet), lahko so pritrjene na toga telesa iz ene (viseča vrv) ali obeh strani (viseči most). Posebno zanimivo področje njene uporabe pa je *plovnost* ali *vzgon*, ki skrbi za obstanek objektov na gladini tekočine. Ta je po *Arhimedovih* ugotovitvah za neki objekt

enaka teži vode, ki jo ta objekt izpodrine. Torej, če želim, da je neki objekt ploven, torej, da se ne potopi, mora biti njegov volumen oz. količina m^3 prostora, ki ga zavzema, večja ali enaka količini njegove teže v kN^2. Ker pa je računanje volumna lahko pri kompleksnih predmetih zapletena operacija, se bomo zadovoljili z njihovo posplošitvijo v obliki *mejne škatle (bounding box)*, ki v celoti zaobjame model v kocko oz. kvader. Sama sila je sorazmerna z globino, do katere se potopi objekt, kar lahko imitiramo s proporcionalnim raztegom vzmeti. V primerih, kjer uporabimo tovrstne strategije, se vzmeti obnesejo, vendar le, če objekt ni v celoti potopljen.

[2]Težo objekta na Zemlji dobimo z izračunom $g * m$, kjer je $g(ms^{-1})$ gravitacijska sila Zemlje, $m(kg)$ pa masa objekta



Slika 4.1: Razteg/skrčenje spiralne vzmeti in količina sile, ki se pri tem proizvede [18]

V primeru, da objekt v celoti potopimo, torej, ga postavimo pod gladino, pa se vzgon ne bo več višal, saj se z večanjem globine potopa ne veča tudi količina izpodrinjene vode (ali kake druge tekočine). Ker imamo za zdaj dela z delci brez mase, ne moremo vedeti kdaj se objekt v celoti potopi, tako da moramo za posamezen objekt določiti globino, ko se to zgodi. Skratka, srečamo se lahko s tremi sledečimi primeri:

$$f(d) = \begin{cases} 0, & \text{če } d \leq 0 \\ \upsilon\rho, & \text{če } d \geq 1 \\ d\upsilon\rho, & \text{sicer} \end{cases} \tag{4.5}$$

Torej, objekt je lahko popolnoma izven vode, objekt je popolnoma potopljen ali pa je delno potopljen. d je delež objekta, ki je potopljen, υ volumen objekta, ρ pa gostota tekočine. Samo potopljenost objekta pa dobimo na naslednji način:

$$d = \frac{y_o - y_w - s}{2s} \tag{4.6}$$

kjer y_o in y_w predstavljata y koordinato sveta[3] pozicije objekta in pa gladine tekočine (zadevo smo si poenostavili s skalarjem y namesto uporabe vektorja, ker predpostavljamo delovanje vzgona navzgor).

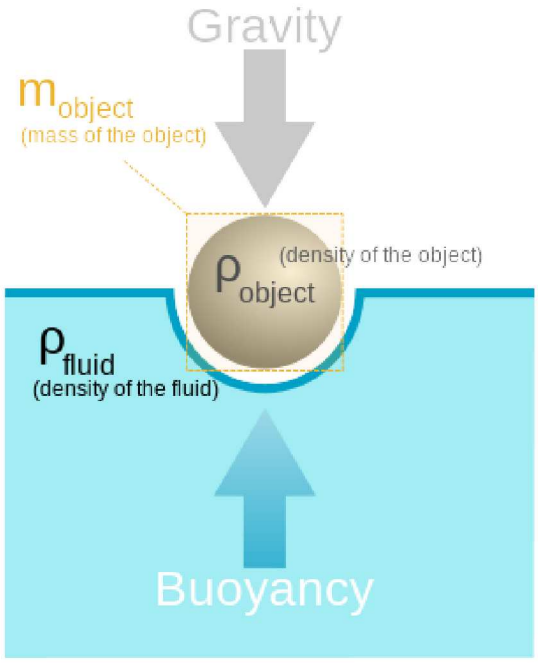
4.2 Trde vzmeti

Za logiko obravnavanja trkov med objekti bi vsekakor lahko uporabili ravnokar predstavljeno obliko vzmeti, ki se uporablja pri plovnosti. Ta način se je obnesel v številnih komercialnih igrah, vendar pa so ti trki videti zelo spužvasto, saj dejansko določimo za posamezne objekte meje (ki so zelo majhne seveda), do katere penetrirajo drug v drugega, in se nato od njega odbijejo v nasprotno smer. Metoda je imenovana *kazenska metoda (penalty method)*, in je zaradi številnih problemov, med drugim tudi številskimi, ne bomo uporabili. Velik

[3]Koordinate globalnega koordinatnega sistema igre oz. pogona.

problem je tudi v tem, da je treba ob vsaki posodobitvi pozicije vzmeti nad njo izvesti različno silo, saj se ta s časom spreminja, glede na njeno raztegnjenost oz. skrčenost (vzmetna sila se skozi časovni interval manjša, saj se vzmet giblje proti njeni mirujoči dolžini).

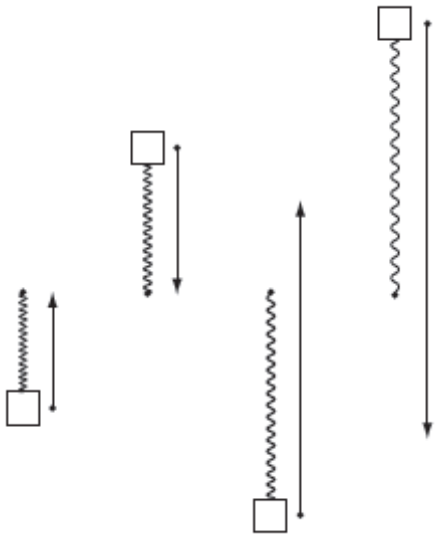
To je zelo zahteven problem in če ga ne rešimo, se bo ob predolgem intervalu krčenja oz. raztezanja vzmeti le-ta lahko z raztegovanjem približevala neskončnosti, namesto da bi s časom njeno gibanje pojenjalo. Ta problem narašča z višanjem *konstante vzmeti*, torej vzmeti, ki imajo hitre in močne odboje, zlasti ko se vzmet giblje hitreje kot pa znaša pretečen čas med posameznimi izrisanimi sličicami, pri čemer vzmet popolnoma ponori (kar se pa zgodi v skoraj vseh primerih, če to strategijo uporabimo za razreševanje trkov med objekti). Zadevo lahko rešimo z izračunom povprečne sile v celotnem časovnem intervalu premikanja vzmeti in predvidevanjem spremembe v sili,



Slika 4.2: Potop objekta in posledičen izpodriv tekočine, ki povzroči vzgon [33]

namesto računanja za vsako izrisano sličico. Temu pravimo *implicitne vzmeti (implicit springs)*, tovrstni fizikalni pogoni pa se imenujejo *implicitni* ali *semi implicitni*.

Uporabne so predvsem v primerih, ko na njih ne delujejo druge sile, tako da moramo biti za njihovo učinkovito uporabo previdni (uporabni so na primer za obnašanje las na človeški glavi, kjer že s samim 3D modelom karakterja podamo začetno dolžino las (mirujoča dolžina vzmeti) in nam tako ni potrebno niti upoštevati gravitacijske sile).



Slika 4.3: Trda vzmet skozi čas [12]

Poglavje 5

Trdnostne omejitve

Kot smo videli v prejšnjem poglavju, je razreševanje kontaktotv on kolizij med objekti skoraj nemogoče realizirati z vzmetmi, tako da se bomo raje osredotočili na ti. *trdnostne omejitve (hard constraints)*. Pri tem naj poudarimo, da jih obravnavamo povsem ločeno od sil.

5.1 Razreševanje trkov

V fizikalnem pogonu kot trk med objektoma označujemo tako trk, ki povzroči spremembo v njuni hitrosti in položaju kot tudi le njuno medsebojno dotikanje. Sam trk se v večini primerov izvede izjemno hitro in ga niti ne opazimo, sama objekta pa sta dejansko v preseku. To rešimo tako, da jima na podlagi njunih hitrosti in smeri v času tik pred trkom ta dva parametra ustrezno popravimo tako, da ustrezata obnašanju teh dveh teles po trku in tako efektivno razrešimo kolizijo. Temu pristopu pravimo *razreševanje trkov (collision resolution)*.

5.1.1 Gibanje trkajočih teles

Samo obnašanje oz. natančneje gibanje teles ob trkih je odvisno od njune *približevalne hitrosti (closing velocity)*[1] oz. *oddaljevalne hitrosti (separating velocity)*[2] ki je v prvem primeru pozitivna, v drugem pa negativna:

$$v_c = \vec{v_a} \cdot (\widehat{\vec{s_b} - \vec{s_a}}) + \vec{v_b} \cdot (\widehat{\vec{s_a} - \vec{s_b}}) \tag{5.1}$$

kar je ekvivalentno:

$$v_c = -(\vec{v_a} - \vec{v_b}) \cdot (\widehat{\vec{s_a} - \vec{s_b}}) \tag{5.2}$$

kjer je v_c približevalna hitrost obeh objektov (kot skalar), $\vec{v_a}$ in $\vec{v_b}$ hitrosti objektov, $\vec{s_a}$ in $\vec{s_b}$ njuni poziciji, $\widehat{\ }$ pomeni izračun enotskega vektorja (ker nas zanima le smer vektorja), $\cdot$ pa predstavlja skalarni produkt vektorjev. Pri Enačbi (5.2) nam predznak izraza relativne hitrosti obeh teles $(\vec{v_a} - \vec{v_b})$ pove, ali se telesi med seboj približujeta (predznak $-$), tako kot je v naši enačbi, ali pa se oddaljujeta (predznak $+$). V slednjem primeru ne govorimo o približevalni, ampak oddaljevalni hitrosti v_c.

Ko se telesi zaletita, drug drugemu preprečita nemoteno potovanje, pri čemer se, tudi pri zelo trdnih telesih, malce deformirata oz. natančneje skrčita tako kot vzmeti, pri čemer se nabere sila, ki povzroči njuno ločitev. Obnašanje tega proces je odvisno od materialov teles, kar pa za sabo potegne še številne druge dejavnike, ki pa jih zaradi praktičnosti ne bomo upoštevali[3].

Če imamo *popolnoma elastičen trk (perfectly elastic)*, se *gibalna količina (momentum)*[4] teles med trkom ohrani, kar je razvidno iz enačbe:

$$m_a v_a + m_b v_b = m_a v'_a + m_b v'_b \tag{5.3}$$

kjer sta m_a in m_b masi teles, v_a in v_b hitrosti teles pred trkom (približevalna) ter v'_a in v'_b hitrosti teles po trku (oddaljevalna). Tukaj je upoštevana naslednja

[1] Hitrost, s katero objekta potujeta drug proti drugemu

[2] Hitrost, s katero objekta potujeta stran drug od drugega

[3] In jih tudi ne moremo upoštevati tako, da bi nam bolj koristili kot škodili

[4] `http://www.physicsclassroom.com/class/momentum/u4l1a.cfm`

predpostavka: *Če na sistem ne deluje nobena zunanja sila ali je rezultanta zunanjih sil enaka nič, se gibalna količina sistema ohranja*[5], kar pa na Zemlji ne velja, tako da moramo zgornjo enačbo prilagoditi.

Zgornja enačba nam pove le skupno hitrost pred in po trku, informacije o hitrosti pred/po trku posameznega objekta pa ne pridobimo. Ti sta povezani med seboj preko formule:

$$v_s' = -cv_s \tag{5.4}$$

kjer sta v_s in v_s' hitrosti pred (približevalna) in po (oddaljevalna) trku, c pa je *koeficient povrnitve (coefficient of restitution).* Ta določa hitrost ločitve teles po trku. Kot smo že omenili, je ta hitrost odvisna od materialov teles, vključenih v trk. c pa je odvisen od kombinacije materialov teles, ki so udeleženi v trku in tako zagotovi pravo razreševanje trkov. Pri $c = 1$ se bo med njima zgodil popolni odboj brez izgube hitrosti, pri $c = 0$ pa pa se telesi ne bosta odbili, ampak se bosta dotaknili in tako potovali skupaj (njuna ničelna oddaljevalna hitrost bo 0). To nam pove, da se gibalna količina ohranja, ne glede na vrednost koeficient povrnitve in bo Enačba (5.3) še vedno veljala. Tako lahko s pomočjo enakosti Enačbe (5.4) in Enačbe (5.3) izračunamo oddaljevalni hitrosti obeh teles v_a in v_b'

V igrah je po navadi več nepremičnih teles kot pa premikajočih. Tem bi lahko določili neskončno maso, kot smo že omenili, vendar je tak ukrep povsem odveč. Boljša rešitev je, da od objekt, ki ga fizikalno ne simuliramo, vzamemo normalo na tistem delu tega objekta, kjer se je zgodil trk z nekim drugim simuliranim objektom in ga uporabimo v posplošeni obliki Enačbe (5.2):

$$v_c = (\vec{v_a} - \vec{v_b}) \cdot \widehat{vecn} \tag{5.5}$$

kjer je *vecn* normala nepremičnega objekta, v tem primeru poimenovana *normala trka ali kontakta (collision normal oz. contact normal).* Enako kot pri Enačbi (5.2) zgornja enačba predstavlja oddaljevalno hitrost, negirana pa približevalno. Predznak *vecn* pa je indikator smeri trka, ki je za izračun odda-

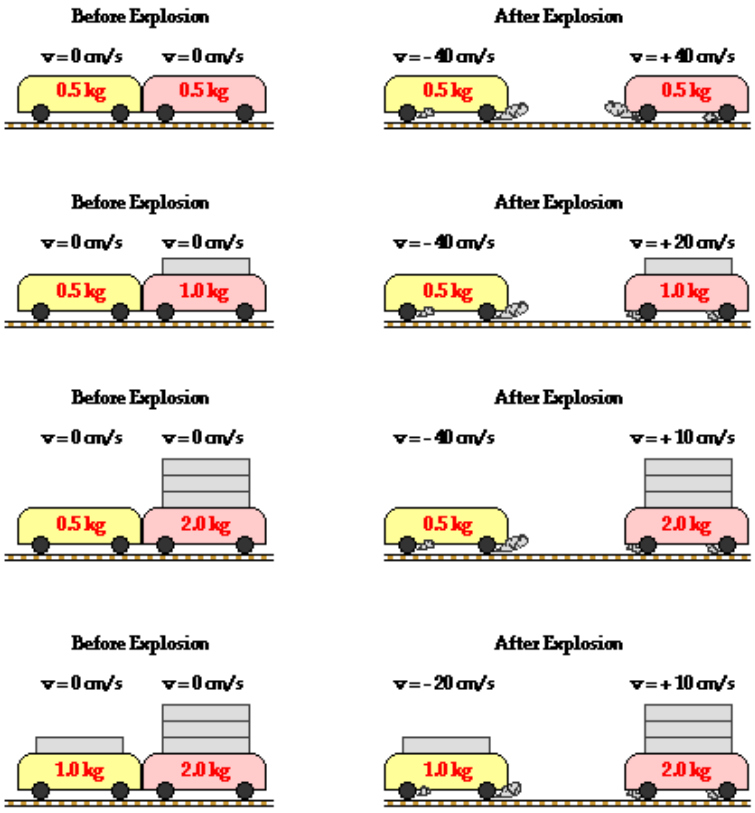
[5] `http://www.nauk.si/materials/4392/out/#state=2`

ljevalne hitrosti prvega telesa (v našem primeru premikajočega) pozitiven in drugega (v našem primeru nepremičnega) negativen.

5.1.2 Sunek sile

Pri razreševanju trkov je vsa kar moramo narediti to, da spremenimo hitrosti tkrajočlih se objektov[6]. Omenili smo, da bomo spreminjali položaj in hitrost teles le preko pospeškov, da bomo dosegli njihovo pravilno obnašanje. Vendar potrebujemo za spremembo hitrosti na določeno vrednost pospešek aplicirati na objekt dlje časa, pri trku pa je sprememba hitrosti praktično takojšnja.

[6]Vektor hitrosti vsebuje tako hitrost kot smer gibanja.



Slika 5.1: Obnašanje dveh teles pred in po trku pri različnih masah [17]

Podobno kot lahko s takojšnjo spremembo sile takoj spremenimo pospešek, želimo takojšnjo spremembo hitrosti. To predstavlja *sunek sile (impulse)*[7]:

$$I = m\Delta v \tag{5.6}$$

kjer je v našeem primeru Δv sprememba v hitrosti telesa pred in po trku. Delovanje vseh sunkov sil na objekt lahko po *D'Alembert*-ovem principu apliciramo tudi na sunek sil in tako seštejemo vse sunke sil, ki delujejo na neki objekt in to vrednost nato uporabimo. Vendar naj poudarimo, da s sunkom sile le spreminjamo hitrost in je ne nastavljamo[8]. Torej bo hitrost po vključitvi vseh sunkov sil nad objektom:

$$v' = v + \frac{1}{m}\sum_{i=1}^{n} I_i \tag{5.7}$$

Ampak za naše potrebe bo bolj praktična uporaba sunkov sil posamično, saj bo v veliki večini primerov nad objekti naenkrat deloval le en sunek sil, in sicer v trenutku trka (v zelo redkih primerih bo objekt trčil na več mestih naenkrat), deloval pa bo nad obema objektoma, udeleženima v trk:

$$v' = v + \frac{I}{m} \tag{5.8}$$

Sunek sile je pravzaprav delovanje sile skozi čas, ali:

$$I = ft \tag{5.9}$$

5.2 Detekcija trkov

Preden lahko razrešimo trke, jih moramo najprej poiskati. Za to je potrebna informacija o obliki objektov, ki je pri delcih nismo potrebovali (ker nimajo velikosti). Ugotoviti moramo torej, kje in kdaj se posamezna objekta dotakneta ali prekrivata in nato izračunati normalo kontakta. Algoritmov za to je ogromno. Nekateri celo predvidevajo prihodnje trke glede na gibanje posameznih teles, drugi pa le preverjajo pare teles ali se sekajo/dotikajo.

[7]Sunek sile deluje v smeri trka (z ustreznim predznakom obeh udeleženih teles v trku).

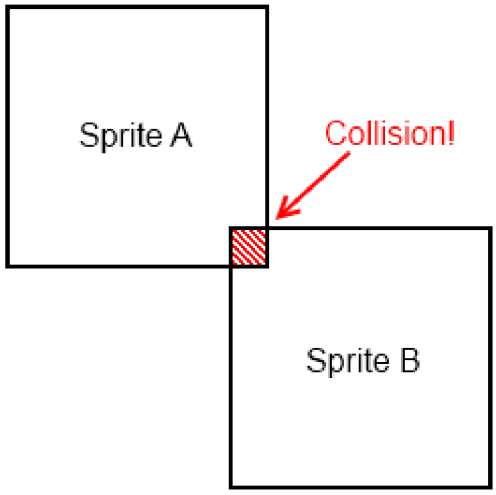
[8]Telo bo imelo hitrost tudi, ko nanj ne deluje nobena sile in/ali sunek sile.

5.3 Prekrivanje

Kot smo omenili, lahko zaznamo trk, takoj ko se telesi med seboj dotakneta, v večini primerov pa se to zgodi malce pozneje, ko se telesi že prekrivata. Rekli smo, da trke rešujemo s spremembo hitrosti. Težava pa nastane, če imamo zelo nizek koeficient povrnitve, ki lahko povzroči, da se telesi ne oddaljita dovolj drug od drugega, in ostaneta v preseku. To bomo rešili tako, da bomo telesi ob trku najprej ločili v smeri normale kontakta na podlagi *globine penetracije* (prav tako v smeri normale kontakta) enega telesa v drugega, da se bosta le dotikala, in šele nato opravili operacijo razreševanja trka:

$$d_{penetracije} = \begin{cases} 0, & \text{če se telesi dotikata} \\ > 0, & \text{če se telesi prekrivata} \\ < 0, & \text{če telesi nista v stiku} \end{cases} \tag{5.10}$$

kjer je $d_{penetracije}$ dolžina oz. globina penetracije. Odločiti se moramo tudi, kako bomo obe telesi ločili oz. za kolikšen delež globine penetracije bomo



Slika 5.2: Prekrivanje dveh teles v 2D prostoru v primeru njunega trka [19]

premaknili prvi in kolikšen delež bo ostal za premik drugega objekta v stanje, kjer se telesi le dotikata. Torej:

$$d_{penetracije} = \Delta s_a + \Delta s_b \tag{5.11}$$

kjer sta Δs_a in Δs_b razdalji obeh objektov, ki povesta za koliko se morata premakniti v smeri normale kontakta. Ta premika bosta odvisna od razmerij mas obeh teles. Če bosta imela enaki masi, se bosta oba premaknila za polovico globine penetracije, če bo eden z večjo maso, se bo ta premaknil manj kot drugi, če bo pa eden izmed njiju nepremičen (z "neskončno"maso), pa se ta ne bo premaknil, drugi pa se bo za celotno globino penetracije. Določimo ju lahko z enakostjo:

$$m_a \Delta s_a = m_b \Delta s_b = d \tag{5.12}$$

iz katere lahko izrazimo premik posameznega telesa:

$$\Delta s_a = \frac{m_b}{m_a + m_b} d \tag{5.13}$$

$$\Delta s_b = \frac{m_a}{m_a + m_b} d \tag{5.14}$$

Seveda pa moramo vnesti če smer premik, ki je enaka smeri normale kontakta za telo, s katerega perspektive je podana[9] in smeri, ki je negativna tej, za drugo telo:

$$\Delta s_a = \frac{m_b}{m_a + m_b} dn \tag{5.15}$$

$$\Delta s_b = -\frac{m_a}{m_a + m_b} dn \tag{5.16}$$

kjer je n normala kontakta.

5.4 Mirujoča telesa

Takšna implementacija pa ni preveč stabilna. Pri ne preveč hitrih simulacijh se obnese, pri večjih hitrostih pa nastane problem, namreč lahko se zgodi,

[9]Normala kontakta je vedno podana iz perspektive enega izmed teles, pri drugem pa moramo upoštevati negativo te smeri.

da gresta telesi popolnoma eden skozi drugega in v tem primeru trka sploh ne bomo zaznali. V vsakem primeru pa imamo problem tudi pri mirujočih objektih ali pa, če imajo ti zelo nizko hitrost in so v stiku z nepremičnihobjekti, kjer bodo vibrirali in občasno skakali. To se zgodi zato, ker ko zaznamo trk med premikajočim in mirujočim objektom, kot smo spoznali, mu potem, ko ga premaknemo, v primeru, da je v preseku, spremenimo hitrost, da trk razrešimo in čeprav je ta zelo majhna, je kljub temu igralcu opazna (primer predmeta na Zemlji, kjer nanj deluje gravitacija in mu posledično vsako izrisano sličico doda hitrost). Daljši, kot je čas med posameznima izrisanima sličicama, bolj bo telo poskočilo[10].

V resnici bi na telo, ki bi prišlo v stik z nepremičnim objektom, delovala sila v smeri normale kontakte nepremičnega telesa. Ta bi telo potiskala nazaj in efektivno izničila pospešek v smeri normale kontakta oz. gravitacijo, če vzamemo za primer površino Zemlje. Zadeva pa je problematična pri kompleksnejših telesih, ki imajo lahko mnogo točk, kjer se dotikajo z nepremičnim objektom. V takem primeru je zelo težko obravnavati vse dotike pravilno oz. na način, da se bo telo kot celota obnašalo pravilno.

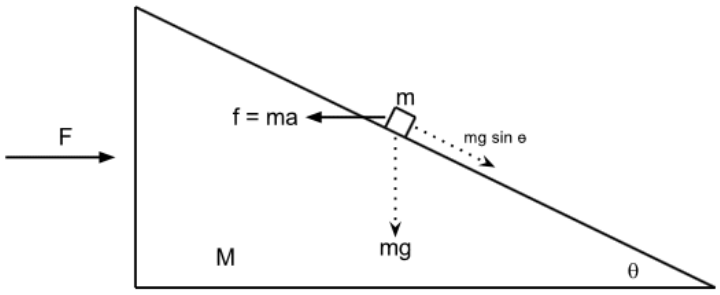
Zadevo lahko poenostavimo tako, da ugotovimo, ali gre dejansko za trk, kjer se telo odbije od drugega, ali pa telo le počiva na nekem drugem telesu. Ali telo počiva ugotovimo tako, da prepoznamo, kdaj ima hitrost, ki bi lahko bila le posledica sil, ki delujejo nanj le za čas ene izrisane sličice. Za primer gravitacije bi izračunali hitrost objekta, če bi nanj delovala le gravitacija in jo primerjali s hitrostjo objekta - če bi bila ta manjša ali enaka[11] izračunani, pomeni, da je telo prejšnjo izrisano sličico mirovalo in gre zatorej najverjetneje za počivajoči objekt (v tem primeru mu ne prištejemo hitrost, ki jo proizvede pospešek oz. gravitacija), sicer pa za trkajoč. V primeru počivajočega telesa temu pripišemo sunek sile, ki upošteva ločevalno hitrost enako 0, s čimer

[10]Torej, če se število sličic na sekundo upočasni, bodo vibracije oz. poskoki toliko bolj vidni.

[11]Lahko bi dopustili, da bi bila lahko tudi malce večja, zaradi zaokroževalnih napak

telo ohranimo na mestu oz., poskrbimo da se dotikata. To ponavljamo vsako izrisano sličico, vse dokler je telo smatrano za mirujoče. Odvijajo se torej minimalni trki in fizikalne pogone, ki jih uporabljajo, imenujemo *pogoni z mikro trki (microcollision engine).*

Naletimo lahko na primere, pri katerih telesi mirujeta glede drug na drugega v eni smeri, v drugi smeri pa se premikata (na primer drsenje telesa na drugem telesu). To pomeni, da moramo v takih primerih uporabiti njuno relativno in ne absolutne hitrosti; sam izračun pospeška in hitrosti teles izračunamo samo za smer normale kontakta. To pomeni, da izračunamo hitrost telesa v tej smeri in preverimo, tako kot smo že pokazali prej, ali bi jo lahko povzročil pospešek v isti smeri. V tem primeru se telesi dotikata, morebitne relativne hitrosti v drugih smereh (na primer hitrost drsenja telesa po drugem telesu) pa nas za namene dotikov teles ne zanimajo.



Slika 5.3: Sile delujoče na telo na klančini [20]

5.5 Vrstni red razreševanja trkov

Spoznali smo, da moramo pri obravnavi kontaktov med objekti upoštevati naslednje tri situacije: objekta se lahko le dotikata, lahko se drug od drugega odbijeta, pri tem pa lahko zaradi časovnega vzorčenja (simulacijo obravnavamo

za vsako izrisano sličico) drug v drugega penetrirata. Pri tem pa je pomemben vrstni red razreševanja kontaktov. Namreč s tem, ko recimo razrešimo enega, smo jih lahko posledično poleg njega še več ali pa smo jih nehote povzročili še več. Poleg tega lahko pri obsežnejših simulacijah pride do primera, ko bo iskanje rešitve trajalo predolgo, ali pa se sploh ne bo končalo. Kakorkoli že, potrebno je pametno upravljanje.

Ena izmed rešitev je takšna, da razrešimo najprej najbolj problematične kontakte. Kot najbolj problematične interpretiramo tiste z najmanjšo ločevalno hitrostjo, torej, najbolj negativno. To utemeljimo z dejstvom, da bodo ti kontakti najbolj vplivali na obnašanje objekta oz. simulacije in posledično na realizem. Ta rešitev je učinkovita hkrati pa zelo dobro oponaša realistično obnašanje v takšnih primerih.

Torej, lahko bi kontakte razvrstili od najbolj do najmanj problematičnih in bi razrešili najbolj problematičen kontakt, bi prešli na obravnavanje naslednjega najbolj problematičnega kontakta. Pri tem pase lahko zgodi, kot smo omenili, da se pri nekaterih kontaktih le-ta odpravi ali pa se na novo povzroči. To bomo rešili tako, da bomo po vsakem razreševanju najbolj problematičnega kontakta, vsem kontaktom preračunali ločevalno hitrost in jih ponovno razvrstili od najbolj do najmanj problematičnega. To bi lahko ponavljali dokler ne razrešimo vseh kontaktov, ker pa se nočemo zaciklati (zlasti pogosta težava pri simulacijah, ki vključujejo trenje) pri kompleksnejših simulacijah, bomo postavili mejo. Ta naj bo vsaj enaka številu kontaktov, da lahko vsi vsaj potencialno pridejo na vrsto enkrat, z večanjem meje pa lahko izboljšamo rezultat, vendar za ceno daljšega računanja simulacije.

V zgornji obravnavi pa nismo vzeli v upoštev morebitnih presekov med objekti oz. medsebojne penetracije. Te bomo obravnavali posebej in pri tem bomo ubrali podoben algoritem. Pri tem bomo podobno kot pri ločevalni hitrosti poiskali najbolj problematična objekta, ki sta v tej situaciji tista z najglobljim presekom. Da pa ne bi vedno znova zaznavali trke, bomo beležili spremembe v globini presekov posameznih objektov. Tudi tukaj postavimo

algoritmu limito iskanja.

Ta limita je lahko v obeh primerih enaka ali se razlikuje. V grobem pa se število kontaktov in penetracij ujema, tudi pri obsežnejših simulacijah, tako da je potreba po ločeni limiti povsem odveč. Ta dva algoritma smo ločili zato, da dobimo optimalne rezultate, namreč vrstna reda razvrščanja kontaktov in presekov od najbolj do najmanj problematičnih sta lahko povsem različna in bi s tem izgubili stopnjo realizma. Tovrstno preračunavanje vrednosti je časovno zelo zamudno. Ob prisotnosti velike količine trkov lahko zavzema večinski delež procesorskega časa. K sreči, v večini praktičnih primerov izračuni oz. trki ne bodo vplivali drug na drugega.

V resničnem svetu se kontakti odvijajo nekaj časa in niso instantni. Kontakti lahko skupno delujejo na telesa, samo obnašanje posameznega kontakta pa je lahko bolj kompleksno. Vsak kontakt se po navadi zgodi v svojem času, za še večjo mero resničnosti pa bi morali počivajoče objekte obravnavati hkrati. V inženirskem svetu se fizikalna simulacija posodablja večkratno na posamezno izrisano sličico, v primerjavi z enkratnim posodabljanjem v našem primeru. V takšnih simulatorjih se dejansko razčleni izrisana sličica na več delov in sicer glede na čas posameznih kolizij. Ko naletimo na prvi kontakt izmed vseh prisotnih v intervalu med zaporednima sličicama, le-tega razrešimo in nato znova poženemo algoritem zaznavanja trkov. To je nadvse požrešno in povsem neprimerno za veliko večino iger. Tudi tu pa se srečamo z numeričnimi napakami, ki nam lahko zagodijo do te mere, da se simulacija zacikla. Vzrok temu je sočasna pojavitev vrste trkov (katerih razreševanje lahko povzroči nove). Dobra stran tega pristopa pa je, da se lahko znebi logike za sunke sil, saj na ta način ugotovimo točen čas dotika posameznih teles in tako do prekrivanja sploh ne pride.

5.6 Povezave teles

Pridobljeno znanje lahko uporabimo pri povezavah med telesi. Ena izmed takšnih so gotovo vrvi. Telesi, ki sta povezani s prožno vrvjo, ta nanju ne vpliva, vse dokler ni vrv popolnoma napeta. Obnašanje v tej situaciji pa je nato odvisno od koeficienta povrnitve posameznih teles, pripetih na konca vrvi. Pri nizkem koeficientu bosta bolj obmirovala, z njegovim večanjem pa se bosta vse bolj agresivno odbila drug proti drugemu. V slednjem primeru interpretiramo zadevo kot kontakt, ki smo jih že omenili, pri čemer negiramo normalo kontakta, da dobimo ustrezno obnašanje vrvi. Torej je logika za trke potrebno le ob iztegnjeni vrvi in ne prej. Globina prekrivanja pa v tem primeru pomeni, za koliko se je vrv raztegnila čez mejo njene maksimalne dolžine, oz. za koliko je to dolžino presegla razdalja med telesi, pripetimi na obeh koncih vrvi.

Sedaj lahko simuliramo tudi neupogljive palice oz. drogove. Gre v bistvu za nadgradnjo vrvi v kombinaciji s trki. Glavna vidna razlika je v tem, da sta telesi na obeh koncih droga konstanto na isti medsebojni razdalji. To bomo najenostavnejše dosegli tako, da bomo popolnoma eliminirali njuno odbojnost - njun koeficient povrnitve nastavimo na 0, s čimer pridobimo relativno hitrost med telesi prav tako enako 0. Gre pravzaprav za enak princip kot pri mirujočih telesih, le da se tukaj telesi ne stikata, ampak je med njima prisotna neka distanca. Za razliko od vrvi moramo tu nad telesi izvesti kontakt, ki ju bodisi združi, če se oddaljita, bodisi loči, če se približata in na ta način iz sličice v sličico poskrbi za pravilno obnašanje droga. Zadevo bi lahko reševali le z razreševanjem kontakta enega izmed pritrjenih teles na drog, vendar bi bil rezultat vibrirajoči drog. Zgodil bi se to, da bi se neko izrisano sličico drog malce skrajšal, spet drugo podaljšal, pod vplivom upora pa bi ga za manjše razdalje premetavalo sem ter tja. Da se izognemo tej posledici razreševanja, nujno izvedemo kontakt nad obemi telesi hkrati (v istem ciklu). Pri tem pa potrebujemo tudi zadostno število iteracij razreševanja, ki raste s kompleksnostjo

uporabe drogov, sicer je tresenje zopet vidno. Opisano logiko lahko izpustimo, če je ločevalna hitrost obeh teles že enaka 0 oz. če ne pride do "prekrivanja"(mišljeno v istem smislu kot pri vrveh globina prekrivanja).

Poglavje 6

Rotacije

Naš cilj je fizikalni pogon s togimi telesi, za to pa moramo realizirati rotacije, poleg linearnega gibanja. Pri tem bomo morali poleg vključitve novih pristopov tudi prilagoditi veliko dosedanjih rešitev.

6.1 2D rotacije

Zaradi kompleksnosti problema si bomo najprej ogledali rotacije v 2D prostoru, iz česar bomo izhajali pri njihovi realizaciji v 3D prostoru.

Kot oz. usmeritev nekega telesa je podana relativno glede na njegovo usmerjenost v prostoru (podobno kot je pozicija podana glede na neko fiksno točko, npr. koordinatno izhodišče), *kotna hitrost (angular velocity)* telesa pa je določena, kot prvi odvod zasuka telesa glede na čas (podobno kot hitrost telesa predstavlja prvi odvod njegovega položaja).

6.1.1 Računanje kotov in kotnih hitrosti

Pri računanju moramo biti pazljivi, kajti isti kot lahko predstavimo z različnimi vrednostmi (npr. 30° in 390° predstavljata isti kot). Prav tako lahko količino za kote izrazimo s stopinjami ali radiani. Mi bomo kote predstavili v obliki

vektorjev:

$$\theta = \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} \tag{6.1}$$

kjer je θ znotraj oglatih oklepajev orientacija telesa, predstavljena kot kot v stopinjah, $\vec{\theta}$ pa je vektorska predstavitev te orientacije. V našem primeru (lahko bi izbrali drugačno rešitev) orientacija enaka 0 pomeni usmerjenost telesa v smeri pozitivne smeri koordinatne osi X, s povečanjem kota pa se telo rotira v nasprotni smeri urnega kazalca. S tako obliko predstavitve si olajšamo večino matematičnih operacij kotov in zaobidemo ponavljanje kotnih vrednosti zaradi uporabe kotnih funkcij. V 2D imamo dve, v 3D prostoru pa tri t. i. *stopnje obračanja (degrees of freedom)*. Te so vezane na posamezno os koordinatnega sistema; telo lahko rotiramo (neodvisno) glede na posamično os. Če hočemo zagotoviti, da lahko za nek vektor dejansko ugotovimo njegovo orientacijo, moramo zagotoviti, da je ta vedno enotski[1], sicer ne bo ustrezal gornji enačbi. Ta zahteva je posledica pravil trigonometričnih funkcij (območje enotskega kroga).

Kotna hitrost, ki predstavlja hitrost rotiranja oz. spreminjanja orientacije telesa, pa nima težave, kjer bi lahko neko kotno hitrost lahko prikazali z večmi vrednostmi (π in 3π nista enaki vrednosti, saj ni omejitve gibanja njenega intervala), tako da lahko za njeno predstavitev brez problema uporabimo skalarno vrednost.

6.1.2 Transormacija

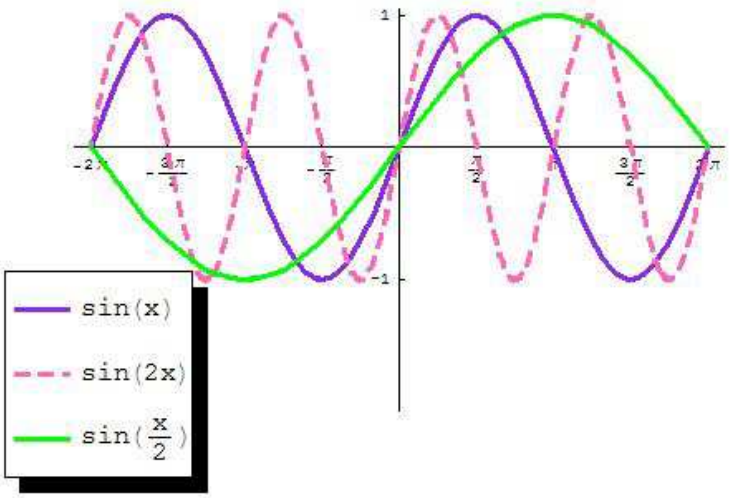
Pri delcih nas je zanimal položaj, rotacije pa niso poznali, saj smo jih interpretirali kot navadne točke. Pri večjih objektih, ki dejansko zavzemajo neki volumen prostora, pa potrebujemo tudi njihovo usmerjenost v prostoru.

V tem primeru se moramo soočiti, kaj sedaj dejansko pomeni njegova pozicija. To lahko določimo kot poljubno točko znotraj globalnega koordinatnega

[1] Z morebitno normalizacijo vektorja; ta se ne glede na dimenzijo vektorja izvede s *Pitagorovim izrekom*.

sistema sveta in predstavlja *lokalni koordinatni sistem* temu specifičnemu telesu. Pomemben je zato, ker določa položaj telesa kot celote; *transformiramo* ga preko te vrednosti. Ta lastnost velja tudi pri delcih, tu pa ima še eno nalogo. Namreč, vse točke, iz katerih je telo sestavljeno, so specificirane relativno glede na to njegovo pozicijo. To je pomembno tedaj, ko želimo transformirati telo. Če želimo telo transformirati (naj gre za translacijo, rotacijo ali skaliranje), moramo poleg njegove pozicije, usmerjenosti in velikosti, pri čemer je vsaka izmed teh lastnosti predstavljena z enim samim vektorjem, ustrezno transformirati še vsa njegova oglišča (torej njegove sestavne dele), ki se izvede sekundarno, relativno glede na transformirane vrednosti telesa kot celote in tako se posledično izvede želena transformacija celotnega telesa.

Pri transformaciji posameznih oglišč telesa je potrebno le vsem vektorjem, ki predstavljajo posamezna oglišča, prišteti vektor pozicije telesa. Pri rotaciji teh pa je malce zapletenejše. Tukaj pa tudi vektorska oblika rotacij ne bo zadoščala za sledeče opravilo, kajti rotirati moramo sam vektor in ne skalar.



Slika 6.1: Sinusna (in tudi druge) kotna funkcija se giblje med intervaloma, ki se preslikata v kot znotraj enotskega kroga (glej Sliko 6.2) [22]

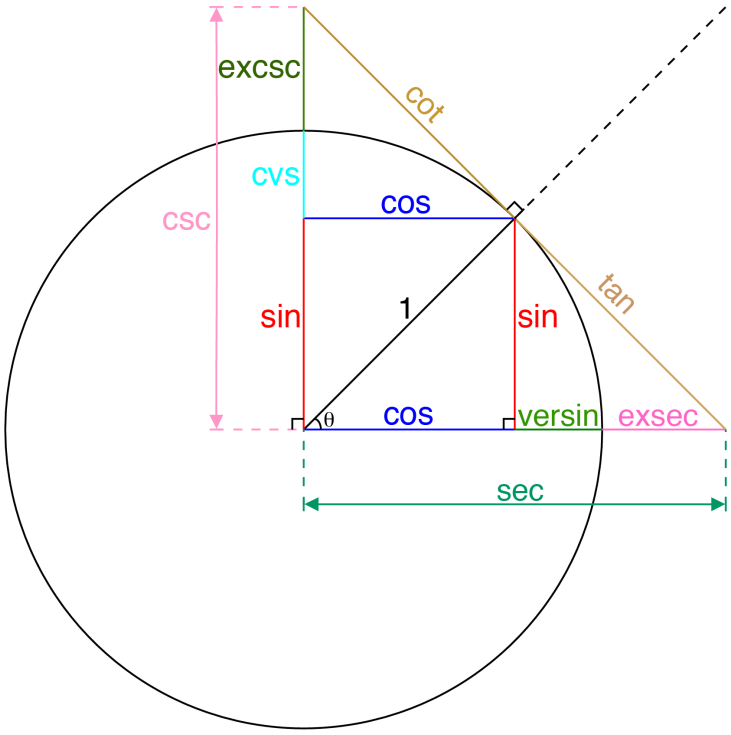
Rotacije bomo morali predstaviti kot matrike:

$$\theta = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{6.2}$$

Gornji matriki pravimo tudi *rotacijska matrika* in nam omogoča rotacijo (relativnih) oglišč telesa glede na rotacijo telesa kot celote. Podobno lahko naredimo za spreminjanje velikosti telesa, pri čemer uporabimo sledečo matriko:

$$S = \begin{bmatrix} v_x & 0 \\ 0 & v_y \end{bmatrix} \tag{6.3}$$

kjer z v_x spremenimo velikost telesa vzdolž osi x, z v_y pa vzdolž osi y.



Slika 6.2: Enotski krog, po obsegu katerega se gibljejo kotne funkcije in tako efektivno predstavljajo vrednosti kotov z intervala od 0° do 360° (ko presežemo 360° smo zopet pri 0°) [21]

6.1.3 Središče mase

Naš pogon bo podpiral izključno toga telesa, kar je tudi praksa pri večini fizikalnih pogonov. Če želimo doseči prožnost togega telesa se pa ta lahko lahko izvede na več načinov, npr. preko delcev ali preko večih togih teles med seboj povezanih z vzmetmi. Kot smo že omenili, imajo toga telesa lahko svojo *izvorno točko* (točka, ki določa položaj celotnega telesa) na poljubnem mestu, ker pa je za uporabo skupaj s fizikalnim pogonom najbolj ugodna točka, ki leži v središču mase telesa, bomo ubrali to pot.

Središče mase ali *središče gravitacije* telesa, ki predstavlja njegovo ravnotežnostno točko oz. nam poda središče telesa glede na razporeditev njegove mase in je pomembna za pravilno obnašanje teles z maso. Omogoča nam, da nad pozicijo telesa izvajamo povsem enake linearne izračune, kot smo jih uporabili nad delci - dejansko središče mase telesa interpretiramo kot navaden delec, telesom moramo le dodati logiko za izvajanje rotacij, ki se izvaja nad tem istim središčem in popolnoma ločeno.

6.2 3D rotacije

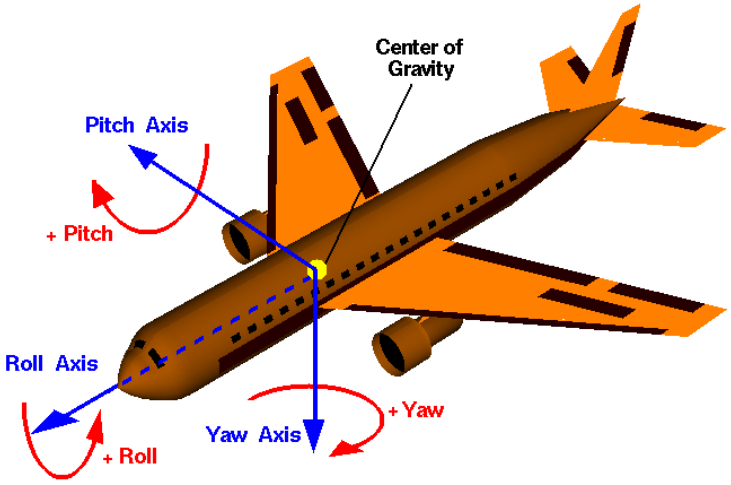
Z rotacijami v 3D prostoru pa se zadeve še močneje zaostrijo in nas za pravilno delovanje rotacije prisilijo v uporabo bolj eksotičnih pristopov iskanja rešitev.

6.2.1 Predstavitve rotacij

6.2.1.1 Eulerjevi koti

Najlažja predstavitev kotov v 3D prostoru je s pomočjo ti. *Eulerjevih kotov* (vpeljal jih je *Leonhard Euler*), kjer je tako kot pri 2D vsak vezan na svojo koordinatno os; rotacijo okoli osi x imenujemo *pitch* (analogno prikimavanju), okoli y *jaw* (analogno odkimavanju) in okoli z *roll* (analogno kotaljenju). Težava pri teh kotih pa je, da ko premaknemo enega, ta vpliva na enega ali celo oba

izmed drugih. Če bi te kote poskušali združiti z vektorskim produktom, bi dobili za različna zaporedja vseh treh vektorjev, ki predstavljajo posamezne kote, različno rešitev, ki bi z naraščanjem rotacije le naraščala.



Slika 6.3: Rotacija telesa okoli osi x (pitch), osi y (jaw) in osi z (roll) [34]

Če pa bi rotacije opravili na način, da posamezna rotacija ne bi vplivala na osi drugi rotacij, bi se srečali pa še z dodatnim problemom - *Kardanska zapora (Gimbal lock)*. Do te pride, ko telo po eni izmed osi rotiramo za 90 glede na njegovo začetno točko, pri čemer se ta os pokrije z eno izmed ostalih in tako dejansko izgubimo eno rotacijsko dimenzijo. Rešimo jo lahko z uporabo drugačnega zaporedja rotacij, vendar se težava le prinese na kombinacijo drugih dveh osi. Dejanskih rešitev obstaja več in so zelo zakomplicirane in nagnjene k programskim napakam, od kombinacije fiksnih osi in osi, ki se gibljejo z rotiranjem telesa, večkratnim rotacijam na isti osi, ugnezdenimi žiroskopi itd.

6.2.1.2 Predstavitev z poravnano osjo

Boljšo rešitev predstavlja *predstavitev s poravnano osjo (axis-angle representation)*. Pri tem uporabimo vektor, ki določi oz. definira os, nad katero nato

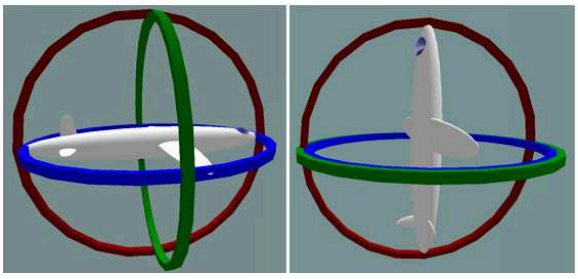
določimo kot. To nam da dodatno *dimenzijo gibanja (degree of freedom)*, ampak ta je potrebna zato, da zagotovimo, da je vektor osi normaliziran, saj predstavlja le smer. Ima enake lastnosti in težave kot predstavitev kota s skalarjem v 2D prostoru (več vrednosti za iste kote).

6.2.1.3 Predstavitev s skalirano osjo

To lahko rešimo s t. i. *predstavitev s skalirano osjo (scaled axis representation)*. Ta nadgradi prej omenjeno predstavitev tako, da normaliziranemu vektorju, ki predstavlja os, podamo dolžino, ki je enaka velikosti kota in tako predstavitev efektivno združimo v en vektor, znebimo se pa tudi problemi z večimi vrednostmi pri istih kotih (interval kota je med 0 in vključno π). Je pa tu problem zahtevnosti računanja, saj je združevanje rotacij precej zahtevno izvesti, kajti rotacije se razlikujejo tako v osi kot v kotu.

6.2.1.4 Rotacijske matrike

Ta predstavitev je najbolj smiselna za uporabo, saj se uporablja v grafičnih karticah pri izračunih za izris zaslona, tako da si z njihovo uporabo prihranimo pretvarjanje v primeru drugačne predstavitve. Nad v matrični obliki zapisanim orientacijam lahko izvajamo matrične operacije, ki pa so dokaj preproste; združitev rotacij v tej predstavitveni obliki je enaka množenju njunih matrik.



Slika 6.4: Kardanska zapora [29]

Sama rotacijska matrika pa je videti sledeče:

$$R = \begin{bmatrix} tx^2 + c & txy + sz & txz - sy \\ txy - sz & ty^2 + c & tyz + sx \\ txz + sy & tyz - sx & tz^2 + x \end{bmatrix} \tag{6.4}$$

pri čemer so x, y in z vrednosti posameznih dimenzij vektorja osi orientacije, s je enak $sin\theta$, c je enak $cos\theta$, t je enak $1 - cos\theta$, θ je pa določeni kot. Zaradi uporabe kotnih funkcij se prav tako kot v 2D primeru znebimo kotov z večimi ekvivalentnimi vrednostmi. Ta oblika predstavitve je dejansko uporabna, vendar pa ima tudi svoj problem. Ta je posledica dejstva, da tri dimenzije gibanja predstavlja s kar devetimi vrednostmi. To nam omogoča, da lahko opravimo raznovrstne transformacije poleg rotacije, kot npr. zrcaljenje in deformacije, vendar pa nas v tem primeru zanima le rotacija in v kombinaciji z matematičnimi napakami pri številih v plavajoči vejici lahko navadna rotacija hitro postane še bilokaj drugega. Zaradi napak moramo zato pogosto preverjati in popravljati vrednosti, da gre še vedno dejansko le za rotacijo, enako kot pri 2D pri predstavitvi s kotnimi funkcijami, kar pa je pri tako velikih metrikah še zamudnejše.

6.2.1.5 Kvaternioni

Zadnje čase najbolj uporabna in dovršena pa je predstavitev z *kvaternion (Quaternions)*, zamisel *William Rowan Hamilton-a*. Ta je po lastnostih podobna rotacijski matriki, razlikuje pa se le v številu uporabljenih vrednosti - uporablja le štiri:

$$\theta = \begin{bmatrix} cos\frac{\theta}{2} \\ xsin\frac{\theta}{2} \\ ysin\frac{\theta}{2} \\ zsin\frac{\theta}{2} \end{bmatrix} \tag{6.5}$$

pri čemer so x, y in z vrednosti posameznih dimenzij vektorja osi orientacije, θ pa določeni kot, ki je relativen na to os. Kvaternion je pravzaprav enak enačbi:

$$q = w + xi + yj + zk \tag{6.6}$$

kjer so ponovno x, y in z vrednosti posameznih dimenzij vektorja osi orientacije, w pa določeni kot nad to osjo. i, j in k pa so različna *imaginarna števila*. Zato jih lahko predstavimo z zapisom v obliki *kompleksnih števil*:

$$\begin{aligned} 1 &= (1,0) \\ i &= (i,0) \\ j &= (0,1) \\ k &= (0,i) \end{aligned} \tag{6.7}$$

Med njimi veljajo naslednje trditve:

- **Ker govorimo o imaginarnih številih, je kvadrat vsakega izmed njih enak -1:**

  $i^2 = j^2 = k^2 = -1$

- **Prav tako je zmnožek vseh enak -1:**

  $ijk = -1$

- **Iz gornjih dveh trditev sledi, da z množenjem poljubnih dveh omenjenih imaginarnih števil dobimo tretje:**

  $ijk = i^2 = j^2 = k^2 \Longrightarrow (ij = k) \wedge (ik = j) \wedge (jk = i)$

- **Kvaternioni niso komutativni**

  $ij = -ji = k$
  $jk = -kj = i$

$$ki = -ik = j$$

Z upoštevanjem pravkar naštetih dejstev lahko kvaternione in z njimi predstavljene rotacije združimo z množenjem na sledeč način:

$$\begin{aligned}(w_1 + x_1 i + y_1 j + z_1 k) \times (w_2 + x_2 i + y_2 j + z_2 k) = \\ (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) + \\ (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) i + \\ (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2) j + \\ (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2) k\end{aligned} \tag{6.8}$$

ali v matrični obliki:

$$\begin{bmatrix} w_1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} \begin{bmatrix} w_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \\ w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2 \\ w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2 \\ w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2 \end{bmatrix} \tag{6.9}$$

Pogoj, da kvaternion predstavlja rotacijo je, da je dolžina vektorja, ki jo predstavlja enaka 1 oz. da je enotski. To pravilo je lahko zelo hitro prekršeno, zaradi zaokroževalnih napak, zato moramo vsake toliko časa kvaternion ponovno normalizirati[2].

Kvaternioni predstavljajo v bistvu enako rešitev, kot smo jo predstavili v 2D prostoru - dodali smo dodatni skalar, s čimer smo dobili dodatno prostorsko dimenzijo, ki pa jo izkoristimo za preprečitev pojava večih predstavitev za iste kote v 3D prostoru oz. natančneje pri kvaternionih, pa tudi zato, da lahko z njimi preprečimo manipulacije drugačne od rotacije. V 2D prostoru smo si definiranje kotov predstavljali z obsegom kroga, tu pa si ga lahko s površino krogle.

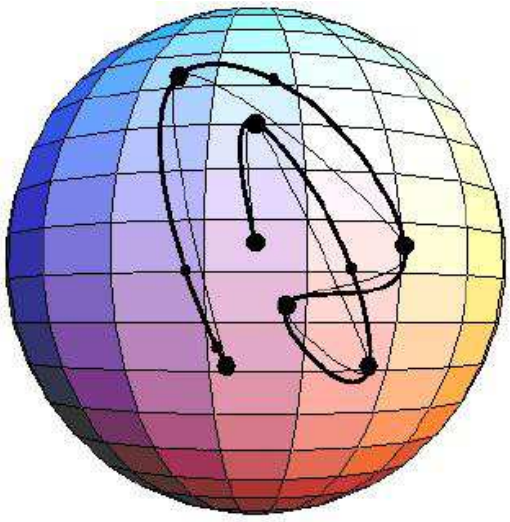
[2]kar opravimo s Pitagorovim izrekom, saj gre pravzaprav za vektor

6.2.1.5.0.1 Pretvorba v transformacijske matrike *Grafični pogoni*, ki izvajajo računske procese znotraj grafične kartice, pa kot smo že omenili pri rotacijskih matrikah, skoraj brez izjeme izvajajo transformacije preko matričnih operacij. To pomeni, da je potrebno transformacije, ki jih predstavimo s kvaternioni, preden jih posredujemo grafičnemu pogonu, pred tem pretvoriti v matrično obliko. Vse kar moramo narediti je, da pridobimo ločeni informaciji za vektor osi in pa kot iz kvaterniona, ki ju združuje, in ju nato vstavimo v že predstavljeno formulo rotacijske matrike, Enačbo (6.4), in tako dobimo:

$$R = \begin{bmatrix} 1-(2y^2+2z^2) & 2xy+2zw & 2xz-2yw \\ 2xy-2zw & 1-(2x^2+2z^2) & 2yz+2xw \\ 2xz+2yw & 2yz-2xw & 1-(2x^2+2y^2) \end{bmatrix} \tag{6.10}$$

pri čemer so w, x, y in z vrednosti komponent kvanterniona.

Naj omenimo, da so matrične operacije zaželene zato, ker nam omogočajo, da lahko z njim vse vrste transformacij obravnavamo enako. To pomeni, da lahko transformacije združujemo z enostavnim množenjem matrik, v primeru



Slika 6.5: Kvaternion za rotacije je vedno enotski vektor, katerega lokacija je omejena na površje enotske krogle [35]

pa da bi posamezno vrsto transformacije predstavili z vektorjem, pa bi morali translacije prištevati, medtem ko bi morali skaliranje in rotiranje množiti. Zgornji rotacijski matriki, ki predstavlja orientacijo telesa, lahko tako enostavno dodamo še podatek o translaciji telesa tako, da vektor (ki imata tri komponente) priključimo tej matriki, kot nov stolpec z desne strani.

Tako pridobljeno *transformacijsko matriko* lahko nato uporabimo za pretvorbo iz lokalnih koordinat v svetovne koordinate in obratno. Prvo pretvorbo dosežemo tako, da pomnožimo transformacijsko matriko z vektorjem, ki predstavlja pozicijo nekega oglišča relativno na telo, kateremu pripada ta matrika. Pri drugi transformaciji je potek enak, razen tega, da moramo transformacijsko matriko pred tem invertirati.

6.3 Kotna hitrost in pospešek

Potrebujemo še način za posodabljanje rotacije telesa odvisno glede na hitrost in pospešek njegovega rotiranja. Ko govorimo o hitrosti, ta nima meje tako kot orientacija oz. nima ponavljajočih se predstavitev istih hitrosti, kar smo videli tudi pri opisu 2D prostora, kjer smo za njegovo predstavitev uporabili le skalarno vrednost. Torej dejansko problem ponavljajočih se predstavitev hitrosti ne obstaja in zatorej lahko za predstavitev *kotne hitrosti* uporabimo kar predstavitev s skalirano osjo, namesto kvaternionov. Predstavimo jo lahko torej z enim vektorjem ali pa množenjem tega istega normaliziranega vektorja in pa kotom oz. v našem primeru frekvenco spremembe kota:

$$\omega = r\vec{a} \tag{6.11}$$

kjer je r frekvenca spremembe kota, po navadi merjena v radianih, $\vec{a}$ enotski vektor smeri, ω pa dobljena kotna hitrost, kjer vsak element tega vektorja pove kotno hitrost telesa okoli posamezne osi. Kotne hitrosti lahko združimo s preprostim vektorskim seštevanjem. Če hočemo narediti kontno hitrost kompatibilno za računanje s kvaternioni, jo moramo spremeniti vanje. To naredimo

enostavno tako, da na mesto zadnjih treh komponent vektorja kvaterniona postavimo vrednosti vektorja kotne hitrosti (gre za vektor s tremi komponentami), na mesto odvečne prve komponente, ki predstavlja velikost kota, pa lahko damo kar 0. Poleg tega, ker ta kvaternion ne predstavlja smeri, ampak skalirno vrednost, ga ne smemo normalizirati:

$$q'_{orientacija} = q_{orientacija} + \frac{\Delta t}{2}\omega q_{orientacija} \tag{6.12}$$

kjer je $q_{orientacija}$ kvaternion orientacije, ω kvaternion kotne hitrosti, Δt pa čas trajanja apliciranja hitrosti (v naši implementaciji gre dejansko za čas pretečen med dvema zaporedno izrisanima sličicama, pri čemer se izračun ponovi, vse dokler ne izvajamo rotacije toliko časa, kolikor se izvaja apliciranje hitrosti nad telesom).

6.3.1 Hitrost posamezne točke telesa

Pomemben podatek nam predstavlja tudi hitrost posameznih oglišč telesa, da lahko omogočimo pravilno procesiranje trkov med telesi. Namreč ko telo kroži, se točke, ki so bolj oddaljene od njegovega središča, vrtijo počasneje kot tiste, ki so bolj oddaljene od njega. Ta hitrost se izračuna na sledeč način:

$$q_{tocka_i} = \omega \times (\overrightarrow{s_{tocka}} - \overrightarrow{s_{telesa}}) + \overrightarrow{v_{telesa}} \tag{6.13}$$

kjer je q_{tocka_i} hitrost neke točke telesa, zapisana v obliki kvaterniona, prav tako ω, ki predstavlja kotno hitrost telesa, $\overrightarrow{s_{tocka}}$ in $\overrightarrow{s_{telesa}}$ sta poziciji telesa in točke (ki je relativna na pozicijo telesa), hitrost posamezne točke telesa pa je odvisna tudi od linearna hitrost telesa $\overrightarrow{v_{telesa}}$.

6.3.2 Kotni pospešek

Kotni pospešek ni pravzaprav nič drugega kot prvi dovod kotne hitrosti po času. Iz tega sledi, da lahko zanj uporabimo popolnoma enako predstavitev

kot pri kotni hitrosti. Poleg tega pa med njima velja enako razmerje kot za hitrost in pospešek v linearnem prostoru. Enačba pa se glasi:

$$\omega' = \omega + \alpha t \tag{6.14}$$

kjer je ω kotna hitrost, α kotni pospešek, t pa čas trajanja apliciranja pospeška (oz. enako kot t pri Enačbi (6.12), le da se tokrat navezuje na kotni pospešek).

Poglavje 7

Toga telesa

Da bo naš fizikalni pogon omogočal resnično toga telesa, bomo morali, kot smo že omenili pri rotacijah, mnogo stvari, ki so bile realizirane nad delci, ponovno implementirati, da bodo delovale nad togimi telesi. Ker smo za izhodiščno točko telesa vzeli središče njegove mase, lahko, kot smo videli, pri togih telesih uporabimo povsem enako logiko linearnega gibanja kot pri delcih. Prilagoditi pa moramo zakone gibanja, ki morajo delovati tudi nad rotacijskimi telesi in pa logiko za sile, da ima ta poleg linearnega tudi rotacijski učinek nad telesom - lahko ga linearno premikajo, hkrati pa tudi rotirajo.

Kot smo spoznali pri delcih, sile delujejo na telo preko pospeška in jim na ta način spreminjajo linearno hitrost ($f = ma$), iz česar pa lahko izrazimo pospešek, ki ga izvaja sila nad telesom ($a = \frac{f}{m}$). Sprememb kotne hitrosti pa diktira namesto sile *navor oz. vrtilni moment (torque)* in namesto mase *vztrajnostni moment (moment of inertia)*:

$$\alpha = \frac{M}{J} \tag{7.1}$$

kjer je α kotni pospešek, ki je pravzaprav predstavljen kot delež kotne hitrosti oz. $\alpha = d\omega$[1], M navor, J pa vztrajnostni moment. To rotacijsko verzijo Newton-ovega 2. zakona gibanja je podal *Leonhard Euler*.

[1]Podobno kot je pospešek pri linearnem gibanju predstavljen kot delež hitrosti.

7.1 Navor

Navor je pravzaprav posebna oblika sile, pri čemer ta namesto linearne privlačnosti ali potiska deluje krožno. Pri pridobivanju navora iz sile (in s tem posledično kotne hitrosti) sta ključna sama velikost sile, ki se izvaja nad telesom in pa razdalja med osjo vrtenja telesa in pa krožnico, po kateri ta sila potuje (torej polmer te krožnice):

$$M = \vec{r} \times \vec{f} \tag{7.2}$$

kjer je M vektor navora, f delujoča sila nad telesom, r predstavlja vektorsko pozicijo (ti. *ročico (axis)*), ki je relativna na oz. poteka od središče mase telesa (in ne skozi katerokoli poljubno izbrano središče telesa), do točke na telesu, kjer se sila izvaja (ti. *prijemališče sile*), $\times$ pa predstavlja vektorski produkt med njima. Iz malce preoblikovane gornje enačbe pa lahko dobimo jakost navora:

$$|M| = |r|\,|F|\,sin\theta = |r|\,|F_{\perp}| \tag{7.3}$$

kjer je θ kot med ročico r in silo F, sila $F_{\perp}$ pa je usmerjena pravokotno na ročico telesa in dejansko proizvede navor, glede na delujočo silo F (glej Sliko 7.1).

V bistvu je navor prisoten pri vsaki sili - če se izvaja sila, se posledično izvaja tudi navor. Izjema je primer, ko vektorja sile f in ročice r kažeta v nasprotno smer, pri čemer sta poravnana drug na drugega. Tedaj bo sila, usmerjena ravno proti središču mase telesa, posledica tega pa je, da na telo ne deluje navor oz. je ta enak 0, medtem ko je linearna sila še vedno prisotna.

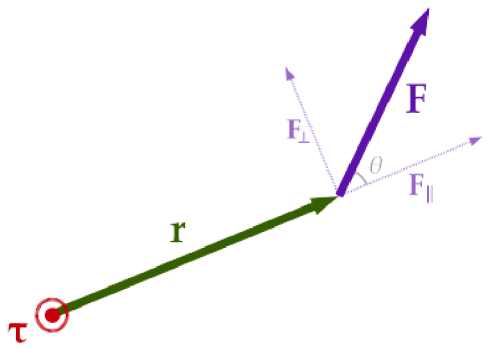
Navor je vedno vezan na neko *rotacijsko os* (glej Sliko 7.3), okoli katere se izvaja oz. rotira telo. Ta os pa mora nujno potekati skozi masno središče telesa, da zagotovimo pravo rotacijo telesa okoli samega sebe. Rotacijsko os lahko poljubno orientirana; torej ni nujno le ena izmed osi baze. Navor lahko

[3] Če bi bil vektor sile F usmerjen na drugo stran glede na ročico r, pa bi se vektor navora raztezal v ekran.

predstavimo pravzaprav kot vektor, ki leži na izbrani rotacijski osi, pri čemer nam njegova usmerjenost (zanjo imamo samo dve možnosti, saj leži navor na rotacijski osi, označimo ju pa s predznakom + ali -) in dolžina povesta, v katero smer deluje oz. vrti telo in kako velik je navor. Torej bi formulo navora Enačbe (7.2) lahko predstavili tudi kot:

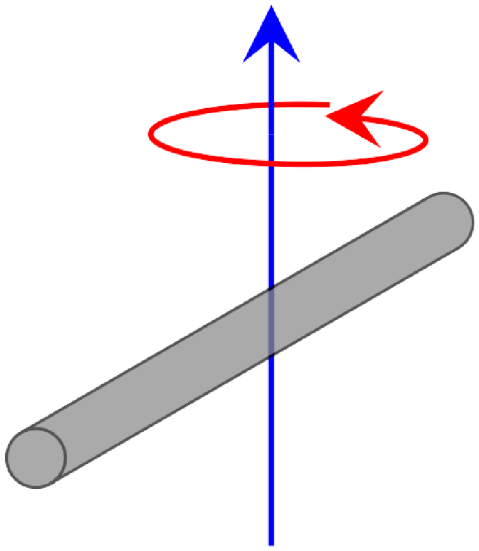
$$M = a\vec{d} \tag{7.4}$$

kjer je $\vec{d}$ enotski vektor, ki poteka skozi izbrano rotacijsko os, a pa je magnituda navora.

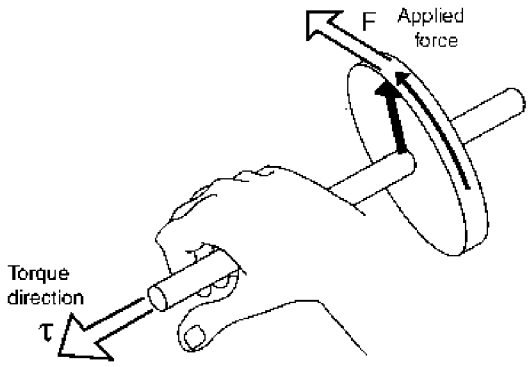


Slika 7.1: Na sliki je τ pravzaprav druga oznaka za navor M, r je ročica navora, F pa sila delujoča nad telesom. Navor je vektor, ki se razteza pravokotno na krožnico navora in poteka skozi središče te krožnice (se razteza izven ekrana[3]), njegova velikost pa pove njegovo magnitudo - predstavlja pravzaprav rotacijsko os, okoli katere telo rotiramo. Če sila F ne deluje pravokotno na ročico navora, se ta sila relativno na velikost kota med njima ustrezno zmanjša. Ta pravokotna sila $F_\perp$, ki je posledica sile F delujoče nad telesom, dejansko proizvede navor [24]

Pri navoru velja *pravilo desne roke*, ki pravi, da če palec kaže v smeri vektorja navora, potem se navor nad telesom izvaja v smeri, v katero so upognjeni ostali prsti roke - glej Sliko 7.3.



Slika 7.2: Rotacijska os telesa, ki poteka skozi njegovo masno središče, okoli katere se telo rotira [36]



Slika 7.3: Pravilo desne roke pri navoru [23]

7.2 Vztrajnostni moment

Vztrajnostni moment telesa nam poda, kako težko je spremeniti rotacijsko hitrost določenega telesa. Prosto gibljivo telo lahko rotiramo poljubno oz. okoli poljubno izbrane osi rotacije. Vztrajnostni moment je odvisno od tega, koko zavrtimo telo; natančneje, odvisna je od mase telesa in od oddaljenosti te mase od izbrane osi rotacije - bolj, ko je ta oddaljena, večji je vztrajnostni moment in posledično telesu težje spreminjamo kotno hitrost (potrebujemo več sile oz. natančneje navora za povečanje in za zmanjšanje kotne hitrosti telesa).

Poleg tega pa velja, da bolj oddaljeno, ko bomo izvajali silo nad telesom oz. bolj, ko bomo oddaljeni od njegove rotacijske osi o (daljša, ko bo ročica navora r), ki bo povzročila navor nad telesom, manj sile oz. navora bo potrebnega, za spremembo kotne hitrosti telesa. In zopet velja tudi obratno; bližje, ko bomo izvajali silo nad telesom oz. bolj, ko bomo bližje njegovi rotacijski osi o (krajša, ko bo ročica navora r), ki bo povzročila navor nad telesom, več sile oz. navora bo potrebnega, za spremembo kotne hitrosti telesa (glej Sliki 7.4 in 7.5).

Če si telo predstavljamo kot skupek večih sestavnih delcev, je enačba vztrajnostnega momenta enaka:

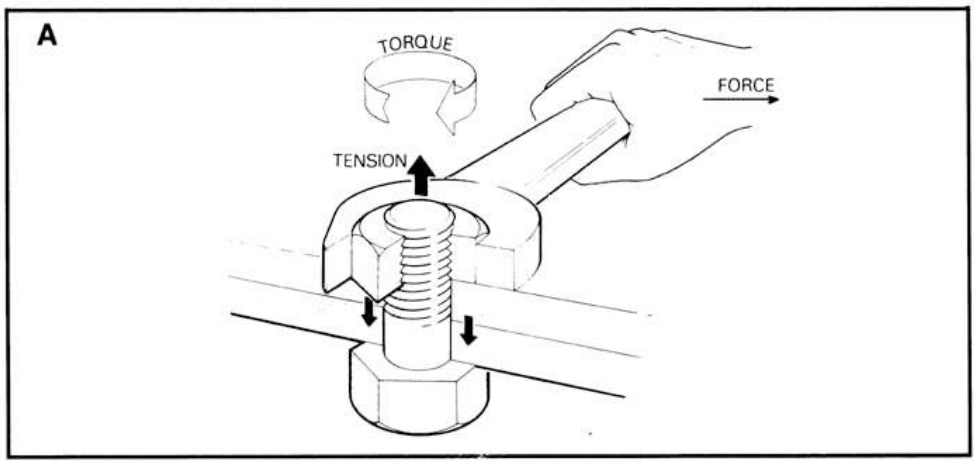
$$J_o = \sum_{i=1}^{n} m_i s_{s_i \to o}^2 \tag{7.5}$$

kjer je n število vseh delcev, ki sestavljajo telo, m_i je masa i-tega delca, J_o je vztrajnostni moment, ki se izvaja nad telesom po krožnici okoli poljubno izbrane osi vrtenja telesa o, na razdalji enaki $s_{s_i \to o}$, ki pomeni razdaljo nekega delca i od prej omenjene osi o. Uporabimo lahko tudi obliko enačbe, ki uporablja integral nad neskončnim številom delcev, torej predstavlja zvezno porazdelitev mase s pomočjo neskončne vsote vseh masnih točk telesa:

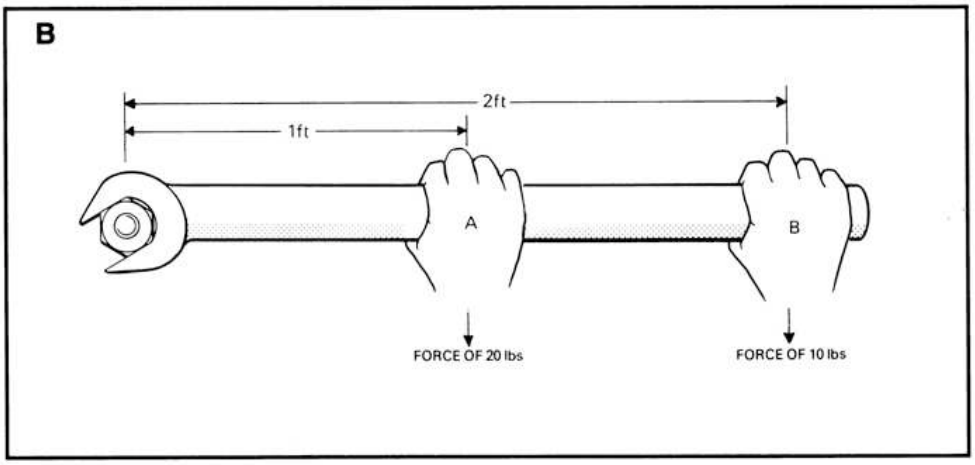
$$J_o = \int_0^n \bar{s}_{s_i \to o}^2 dm \tag{7.6}$$

pri čemer je n število masnih delcev, *dm diferencialna masa (differential mass)*, ki pomeni hitrost spreminjanja mase glede na spremembo radija posameznega

delca oz. njegova oddaljenost od masnega središča telesa in je pravzaprav enako $\rho(\vec{s})dV$, kjer $\rho(\vec{s})$ predstavlja prostorsko porazdelitev gostote mase kot



Slika 7.4: Z rokami ustvarimo silo, ki posledično izvede navor nad matico, in z dovolj veliko silo/navorom se ta odvije/privije (rotira) [37]



Slika 7.5: Daljša, ko je ročica viličastega ključa (predstavlja pravzaprav ročico navora r) in bolj oddaljeno od ustja ključa, ko ga bomo prijeli, lažje bomo lahko privili/odvili (rotirali) matico [32]

funkcijo pozicije nekega delca, dV pa je *diferencialni vlomen (differential volume)*, ki pomeni hitrost spreminjanja volumna glede na spremembo radija posameznega delca oz. njegovo oddaljenost od masnega središča telesa (gre za odvod formule za izračun volumna).

Same delce je dobro razdeliti v diskretne množice in jih nato sešteti, še zlasti ko gre za zelo kompleksno oblikovano telo - na ta način uberemo bližnjico za izračun približka, ki je hitrejši. Pri tem moramo poznati ključno vrednost, to je gostoto ρ, s pomočjo katere lahko izračunamo vztrajnostni moment preprostejših teles in jih nato med sabo seštevamo ali/in odštevamo, dokler ne pridemo do dobrega približka vztrajnostnega momenta kompleksnejšega telesa.

7.2.1 Vztrajnostni tenzor

Vztrajnostni moment v nasprotju z maso pri linearnem gibanju telesa ne moremo predstaviti z eno skalarno predstavitvijo, zato nam gornji enačbi ne podasta dovolj informacij. Vzrok temu je, da je posamezen vztrajnostni moment kot že rečeno vezan na določeno rotacijsko os telesa, ampak na voljo imamo neskončno možnosti za izbiro te osi (glej Sliko 7.1). Ker pa se osredotočamo na toga telesa, je za naše potrebe potrebnih le nekaj izbranih skalarnih vrednosti. Te vrednosti so navedene znotraj *vztrajnostnega tenzorja* ali *matrike mase (inertia tensor/mass matrix)*. Malce drugače, v matrični obliki predstavljena Enačba (7.1), bi potem zgledala tako:

$$\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz} \end{bmatrix} \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix} \tag{7.7}$$

Tenzor pomeni pravzaprav večrazsežno tabelo, ki je pravzaprav posplošena oblika matrike za poljubno dimenzijo. V našem primeru, kjer se nahajamo v 3D prostoru, bo dimenzija tenzorja vedno enaka 2D, tako da bomo imeli

opravka le z 2D tenzorji oz. matrikami. Kot je razvidno iz gornje enačbe, ta vsebuje skupaj devet vrednosti in je kvadratne oblike.

Enako, kot smo pri linearnem gibanju delcev uporabljali inverz mase delca m, bomo pri togih telesih uporabili inverz vztrajnostnega momenta oz. natančneje inverz *vztrajnostnega tenzorja*, ki ga bomo spoznali v nadaljevanju. S takšno predstavitvijo lahko neposredno in s tem hitreje oz. učinkoviteje izračunamo kotni pospešek δ telesa iz Enačbe (7.1), saj nam ni potrebno ob vsakem računanju izvesti inverza matrike vztrajnostnega tenzorja J.

7.2.1.1 Diagonalne vrednosti

V vztrajnostnem tenzorju predstavljajo vrednosti, razporejene po diagonali, vztrajnostne momente, torej J_{xx}, J_{yy} in J_{zz} oz. lahko jih označimo kar J_x, J_y in J_z (bomo videli zakaj pri obravnavi ostalih, nediagonalnih vrednostih). Omenjene diagonalne vrednosti predstavljajo vztrajnostni moment okoli vsake izmed baznih osi izbranega referenčnega oz. lokalnega koordinatnega sistema telesa, z izhodiščem v središčni točki telesa (njegove osi potekajo skozi središče telesa), ki mu omenjeni vztrajnostni tenzor pripada. Ne pozabimo, da mora biti pri vztrajnostnem momentu to središče enako masnemu središču telesa. Vztrajnostni tenzor nam pravzaprav pove, kako je razporejena masa nekega togega telesa glede na izbrani referenčni koordinatni sistem.

Vztrajnostni momenti vztrajnostnega tenzorja so na podlagi Enačbe (7.5) torej enaki:

$$J_x = \sum_{i=1}^{n} m_i(\vec{y}^2_{s_i \to o} + \vec{z}^2_{s_i \to o}) \tag{7.8}$$

$$J_y = \sum_{i=1}^{n} m_i(\vec{x}^2_{s_i \to o} + \vec{z}^2_{s_i \to o}) \tag{7.9}$$

$$J_z = \sum_{i=1}^{n} m_i(\vec{x}^2_{s_i \to o} + \vec{y}^2_{s_i \to o}) \tag{7.10}$$

oz. v obliki integralov (glej Enačbo (7.6); pomagamo si s Pitagorovim izrekom):

$$J_x = \int_0^n (\vec{y}^2_{s_i \to o} + \vec{z}^2_{s_i \to o}) dm \tag{7.11}$$

$$J_y = \int_0^n (\vec{x}^2_{s_i \to o} + \vec{z}^2_{s_i \to o}) dm \tag{7.12}$$

$$J_z = \int_0^n (\vec{x}^2_{s_i \to o} + \vec{y}^2_{s_i \to o}) dm \tag{7.13}$$

Za vsakega izmed vseh treh prej omenjenih vztrajnostnih momentov diagonale velja, da bolj kot je delež mase telesa oddaljen od posamezne osi baze in večji, ko je ta delež, večji bo vztrajnostni moment glede na to določeno bazo. Zato bo za dosego nekega določenega kotnega pospeška potreben večji navor okoli te določene osi. Prav tako velja obratno - večji, ko bo delež masa telesa zbran blizu neke bazne osi, namesto da je od nje bolj oddaljen, manjši navor bo potreben za dosego nekega določenega kotnega pospeška in s tem posledično določene kotne hitrosti. Razporeditev mase se pri deformljivih in/ali gibljivih predmetih lahko spreminja; lep primer so drsalci na ledu, ko se vrtijo - bolj ko se naredijo široke, počasneje se vrtijo, in bolj ko se stisnejo k sebi oz. vzravnajo, bolj je mase zbrana okoli središča in posledično se lahko pri isti količini navora vrtijo hitreje.

Pri telesih, ki so simetrični glede na vse bazne osi[4], torej popolnoma simetrične, in izbrana rotacijska os poteka skozi središče njihove mase, so vse vrednosti vztrajnostnega tenzorja, ki niso na diagonali, enake 0[5]. Razlog tiči v tem, da bo v tem primeru učinek, ki ga povzroči neka masna točka telesa (kot posledica neke sile, npr. gravitacije), ki se v nekem trenutku rotacije nahaja na neki lokaciji, "izničenaž neko drugo masno točko, ki se nahaja na nasprotni strani vsake izmed baznih osi (določenih glede na rotacijsko os telesa), kot se nahaja prva masna točka oz. na lokaciji, ki ustreza prelikavi lokacije prva masne točke skozi lokalno koordinatno izhodišče telesa (določeno z izbrano

[4]V 3D prostoru torej glede na vse tri bazne osi.

[5]V resnici lahko za vsako telo poiščemo bazo oz. skupino osi (ki so seveda pravokotne druga na drugo), pri kateri bo veljal omenjeni vztrajnostni tenzor.

rotacijsko osjo). S to predpostavko o simetričnih telesih lahko poenostavimo matrično Enačbo (7.7) v vektorsko enačbo:

$$\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix} = \begin{bmatrix} J_x\alpha_x \\ J_y\alpha_y \\ J_z\alpha_z \end{bmatrix} \tag{7.14}$$

ki pa so, kot vidimo, povsem enake, razlikujejo se le v tem, da se posamezna navezuje na posamično os. Pri teh enačbah velja enako obnašanje kot pri $f = ma$ pri linearnem gibanju - večja ko je m oz. J, večja sila f oz. večji navor M je potreben, da poženemo telo v linearno oz. krožno gibanje.

Po drugi strani pa vrednosti na diagonali nikoli ne smejo biti 0, saj bi tako zopet predstavili nerealno telo. To bi namreč pomenilo, da bi okoli tistih osi, ki bi v diagonali vztrajnostnega tenzorja imele vrednost nič, kotna hitrost bila enaka neskončno, za nek dani navor (gre pravzaprav za enako ugotovitev kot v primeru, ko ima telo maso enako 0).

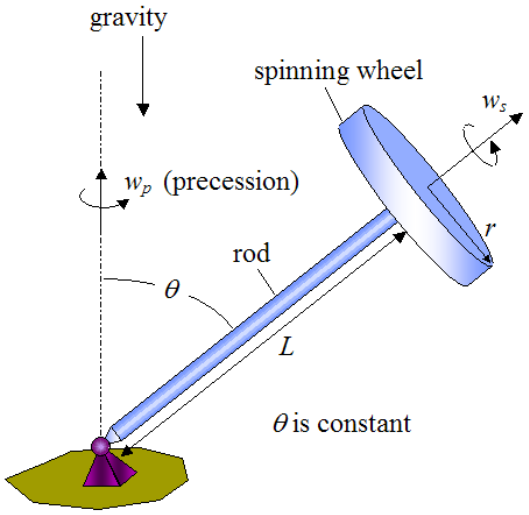
7.2.1.2 Nediagonalne vrednosti

Nek nediagonalni element matrike vztrajnostnega tenzorja, vzemimo za primer J_{zx}, nam pove, za koliko bo določeno telo pospešilo okoli bazne osi x, ko temu telesu apliciramo določen navor okoli bazne osi z[6]. Tem vrednostim pravimo, da so ti. *centrifugalni vztrajnostni momenti oz. vztrajnosti produkti (products of inertia)*. Predstavljajo nam težnjo telesa, da se rotira v drugačni smeri, od tiste smeri, nad katero smo smo izvedli navor nad telesom.

Lep primer za demonstracijo nam predstavlja žiroskop. Žiroskop se upira padcu zaradi gravitacije, in sicer tako, da "prenese"oz. preusmeri silo gravitacije, ki bi povzročila padec žiroskopa (ali natančneje, da "prenese"oz. preusmeri rotacijo, ki jo povzroči gravitacija in ki bi povzročila padec žiroskopa)

[6]Ravno zato smo pri vztrajnostnih momentih (diagonalnih vrednostih) iz predstavitve J_{oo} prešli na predstavitev J_o (kjer je o neka bazna os relativna na izbrano rotacijsko os), kajti gre za isti osi in bi bil takšen opis, kot smo ga podali tukaj, popolnoma odveč in nejasen.

v nasprotni smeri njene prvotne smeri. Ta proces kontrolirajo prej omenjeni centrifugalni vztrajnostni momenti - prenašajo rotacijo telesa iz ene (prvotno izbrane) rotacijske osi na drugo.



Slika 7.6: Žiroskop in njegovo vrtenje, ki preprečuje njegov padec [28]

Kot smo pred kratkim spoznali, bi v primeru simetričnih teles (pri katerih gre izbrana rotacijska os telesa skozi njegovo masno središče) pospešek okoli bazne osi x bil enak 0. Pri nesimetričnih telesih in/ali takšnih, ki nimajo središča v svojem središču mase (natančneje, rotacijska os ne poteka skozi masno središče telesa), pa bi bila ta vrednost neničelna in to bi v praksi dejansko pomenilo, da bi se kotni pospešek nad telesom izvedel po osi, ki ne bi bila enaka tisti, po kateri smo hoteli izvesti kotni pospešek. Ta os bi lahko bila kaka izmed drugih osi baze, v večini primerov pa bi šlo za povsem neko drugo os znotraj (v našem primeru 3D) prostora. Za izračun nediagonalnih elementov matrike

vztrajnostnega tenzorja so dovolj naslednje tri formule:

$$J_{xy} = J_{yx} = \sum_{i=1}^{n} m_i \vec{x}_{s_i \to o} \vec{y}_{s_i \to o} = \sum_{i=1}^{n} m_i (s_i \to o \cdot \hat{x_o})(s_i \to o \cdot \hat{y_o}) \tag{7.15}$$

$$J_{xz} = J_{zx} = \sum_{i=1}^{n} m_i \vec{x}_{s_i \to o} \vec{z}_{s_i \to o} = \sum_{i=1}^{n} m_i (s_i \to o \cdot \hat{x_o})(s_i \to o \cdot \hat{z_o}) \tag{7.16}$$

$$J_{yz} = J_{zy} = \sum_{i=1}^{n} m_i \vec{y}_{s_i \to o} \vec{z}_{s_i \to o} = \sum_{i=1}^{n} m_i (s_i \to o \cdot \hat{y_o})(s_i \to o \cdot \hat{z_o}) \tag{7.17}$$

S pomočjo na novo pridobljenih vrednosti lahko predstavimo vztrajnostni tenzor kot:

$$J = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix} \tag{7.18}$$

Centrifugalni vztrajnostni momenti so lahko, za razliko od vztrajnostnih momentov (ker v enačbi vsebujejo kvadrat nad določeno osjo) tudi ničelni in negativni. Posledično je lahko celotni vztrajnostni moment oz. vsota vztrajnostnih momentov vseh masnih delcev telesa tudi negativna vrednost. Ničelne vrednosti so pogoste pri telesih s čudnimi oblikami, vedno pa so prisotne pri vseh simetričnih telesih, kjer so vse nediagonalne vrednosti vztrajnostnega tenzorja enake 0. Na nediagonalnih vrednostih vztrajnostnega tenzorja, predstavljenega v zgornji enačbi, imamo negativne predznake izključno zato, da ponazorimo, da so lahko centrifugalni vztrajnostni momenti tudi nepozitivni (torej neativni ali enaki 0).

Posledica nepozitivnih vrednosti centrifugalnih vztrajnostnih momentov vztrajnostnega tenzorja je, da se v večini primerov smer vektorja navora M ne ujema s smerjo vektorja kotnega pospeška α. Smeri se ujemata le v primeru, ko so vsi centrifugalni vztrajnostni momenti enaki 0 in vektor navora M sovpada z eno izmed lokalnih baznih osi telesa glede na izbrano rotacijsko os telesa o.

Iz te predstavitve lahko vidimo, da ugotovitev iz Enačbe (7.14) res velja - torej učinek neke masovne točke je lahko izničen, če so posamezni pari centrifugalnih vztrajnostnih momentov, ki se navezujejo na isti dve lokalni bazni osi

telesa glede na rotacijsko os (npr. par J_{xz} in J_{zx}), izenačeni oz. imajo paroma enake vrednosti. Ker pa imata imata lahko posamezna para enako vrednost le pri 0 (saj se pri istih vrednostih izničita,), je edini možni scenarij, da so vse nediagonalne vrednosti oz. vsi centrifugalni vztrajnostni momenti enaki 0. Če velja to za vse masne točke telesa, potem ima celotno telo poponoma enake nediagonalne vrednosti kot njegovi sestavni masni delci - v tem primeru gre za simetrično telo.

V našem fizikalnem pogonu bomo zaradi priročnosti navor navajali v svetovnih koordinatah. Ker pa je vztrajnostni tenzor podan v lokalnem koordinatnem sistemu telesa, ki mu pripada, bomo vsako izrisano sličico izračunali transformacijsko matriko, s katero bomo transformirali vztrajnostni tenzor iz lokalnega v svetovni koordinatni sistem (pri tem lahko zanemarimo položaj telesa, saj nas zanima le njegova usmerjenost oz. smer - tako imamo opravka namesto z matriko dimenzije 4×3, opravka z le rotacijsko matriko dimenzije 3×3). Posledično bo tudi kotni pospešek teles in s tem njihova kotna hitrost prav tako podana v svetovnih koordinatah.

7.2.2 Povezanost navora in sile

Vse navore, ki delujejo na telo, lahko na enak način, kot smo to storili pri silah, ko smo obravnavali linearno gibanje, združimo s pomočjo D'Alembert-ovega principa; torej, vse na telo delujoče navore seštejemo v enega samega, ki ima enak učinek, kot vsi posamezni navori skupaj. Iz tega skupnega navora nato izračunamo samo en kotni pospešek, ki nato vpliva na hitrost telesa.

Pri tem pa je potrebno biti pozoren na dejstvo, da lahko sila povzroči tudi navor nad telesom. To se zgodi v vsakem primeru, razen če je sila usmerjena direktno v masno središče telesa, kot smo že spoznali. Te navore, ki so stranski produkti sil, prav tako prištejemo skupnemu navoru telesa, izračunamo pa jih z že omenjeno Enačbo (7.2). Tako dobimo eno skupno akumulativno vrednost sile in eno skupno akumulativno vrednost navora za posamezno telo.

Nekatere sile pa delujejo izključno na središče mase telesa in v takšnih pri-

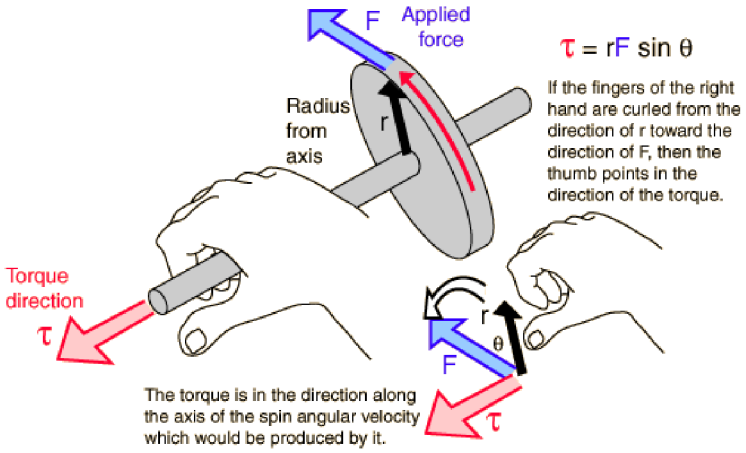
merih se nam ni treba ubadati z navorom, saj takšne sile nikoli ne proizvedejo rotacije telesa. Lep primer predstavlja sila gravitacije.

Torej se srečamo s tremi različnimi vplivi sil[7] nad togimi telesi:

- navor, ki deluje izključno nad masnim središčem telesa in telo rotira okoli njegove osi
- sila, ki deluje na neko določeno masno točko telesa (ponazorjena z vektorjem $\vec{r}$ v Enačbi (7.2) - ročica navora), ki ni enaka masni točki v masnem središču telesa, kot posledico pa povzroči tudi navor (glej Sliko 7.1 - $F_\perp$ je posledični navor sile F)
- sila, ki deluje nad masnim središčem telesa oz. nad masno točko telesa, ki leži v masnem središču taistega telesa

Torej moramo zdaj na vsako izrisano sličico poleg akumulacije sil, sešteti ločeno še vse navore nad posameznim telesom.

[7]Naj ponovno omenimo, da je navor krožna sila oz. gre za silo, ki deluje krožno, in ne linearno, kot sile kot take.



Slika 7.7: Apliciranje sile nad telesom in apliciranje navora nad telesom kot posledica te sile [31]

Kot smo omenili, sile delujejo na posamezni izbrani masni delec telesa (in tako posledično vplivajo na telo kot celoto), torej se lahko pogosto zgodi, da na neke masne delce deluje več sil hkrati, na nekatere pa nobena. Lokacijo teh masnih točk, nad katerimi apliciramo sile, in ki so navedene relativno na središče telesa (mi uporabljamo masno središče), pri tem izrazimo zaradi intuitivnosti kar v svetovnih koordinatah znotraj našega pogona. V svetovnih koordinatah predstavljamo tudi sile, kar smo tudi že povedali. Problem pa lahko nastane pri silah, ki so vezane na telo samo, npr. pri vzmeteh, ki so nanj pritrjene. V tem primeru bi morali ob vsaki izrisani sličici z linearno transformacijo telesa le-to transformacijo izvesti prav tako nad točko, ki predstavlja konec vzmeti, ki je pritrjen na telo (naj poudarimo, da z vzmetjo mislimo silo in ne vzmeti kot telesa). Zaradi tega moramo v takšnih primerih opraviti dodatno transformacijo masne točke telesa iz lokalnih v svetovne koordinate, sicer pa lahko vse izvajamo v svetovnih koordinatah.

7.2.3 Integrator z rotacijami

Integratorju, ki pri delcih posodablja njihovo linearno pozicijo, moramo pri togih telesih poleg posodobitve linearne pozicije telesa (posledica sil) dodati še posodobitev orientacije telesa (posledica navora in posredno sil).

Pri delcih smo vpeljali pojem sile linearnega upora, pri togih delcih pa bomo vpeljali še silo krožnega upora. Ta nam pove, koliko kotne hitrosti oz. natančneje kolikšno količino, odstotek kotne hitrosti izgubi togo telo na vsako pretečeno sekundo kot posledica vpliva upora, ki se izvaja nad telesom, zaradi njegovega vrtenja okoli svoje osi.

Pri delcih smo imeli Enačbo (2.12), pri togih telesih pa imamo skoraj ekvivalentno enačbo:

$$v = vd_k^t + at \tag{7.19}$$

kjer je d_k koeficient krožnega upora.

Poglavje 8

Implementacija fizikalnega pogona

Fizikalni pogon smo implementirali v objektno usmerjenemu programskemu jeziku *C++*, verzija *C++11* (pred tem imenovana *C++0x*). Zanj smo se odločili zaradi že omenjenega poudarka na hitrosti in učinkovitosti, ki jo lahko izkoristimo s tem jezikom. Uporabljamo *MinGW (Minimalist GNU for Windows)* razvojno okolje, ki vključuje prevajalni sistem *GCC (GNU Compiler Collection)*. Za kreacijo in upravljanje z okni uporabljamo knjižico *freeglut*. Kot vmesnik za upravljanje grafike smo uporabili mmedplatformno knjižnico *OpenGL*. Vse našteto spada pod t. i. *odprto kod (open source)*.

8.1 Vektorji

V grafiki (zlasti v 3D, pogosto pa tudi v 2D prostoru) imamo opravka z vektorji. Predstavljajo temeljno osnovo fizikalnega pogona in grafičnih aplikacij nasploh, zato jih potrebujemo za nadaljnjo implementacijo pogona.

Ker smo osredotočeni na 3D prostor, bomo predstavili vektorski razred s tremi v plavajoči vejici podanimi spremenljivkami x, y in z, ki predstavljajo pozicijo neke točke v 3D prostoru kot zamik po vseh baznih oseh koordinatnega

sistema sveta od njegovega izhodišča.

Razred vsebuje vse osnovne operacije nad posameznimi vektorji in med pari vektorjev: seštevanje in odštevanj vektorjev, množenje vektorja s konstanto, množenje dveh vektorjev po komponentah, vektorski produkt (cross product), skalarni produkt (dot product), dolžina vektorja (Pitagorov izrek), normalizacija (dobimo enotski vektor) in raznorazne primerjave (ali je nek vektor daljši do drugega ipd.).

Spoznali smo tudi, da potrebujemo za realizacijo rotacij tudi kvaternione. Pri slednjih potrebujemo dodatno komponento. Te štiri števila si lahko predstavljamo kot koeficiente kompleksnega števila s tremi imaginarnimi deli. Nad tem razredom izvedemo naslednje operacije: normalizacija (pomembna tudi zato, ker le enotski kvaternioni predstavljajo rotacije), množenje in seštevanje kvaternionov ter skalarno prištevanje.

Za predstavitev rotacij po vseh treh oseh bomo implementirali matrike dimenzije $3x3$, ki nam zagotovijo rotiranje teles brez že obravnavanih težav. Uporabimo jih tudi za predstavitev vztrajnostnih tenzorjev.

Ta razred omogoča določanje vztrajnostnih tenzorjev (izkoristimo dejstvo, da so vrednosti zrcalne glede na diagonalo) in še nekaj drugačnih inicializacij, kot na primer vztrajnostni tenzor kocke na podlagi njene mase in dolžine stranice in pa rotacijska matrika na podlagi rotacijskega kvaterniona. Omogoča transformacijo vektorja z matriko (množenje vektorja z matriko, uporabno npr. za apliciranje rotacije nad smernim vektorjem telesa), transponacijo matrik, pridobivanje posamezne osi matrike (stolpca) in vrstice, inverz matrike, seštevanje in množenje matrik (slednje omogoča združevanje oz. akumulacijo večih transformacij v eno samo matriko), množenje in seštevanje matrik po komponentah s skalarjem.

Zaradi priročnosti in hitrosti izvajanja bomo vključili tudi matrike z 12 elementi (dimenzije $3x4$), ki združujejo rotacijsko matriko (dimenzije $3x3$) in pozicijski vektor v eno samo matriko. Pri dejanskem računanju pa vključimo dodatno, četrto vrstico, ki je vedno enaka vektorju $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ in predstavlja

homogeno matriko. Vektorju, nad katerim izvajamo rotacijo in translacijo, pa dodamo *homogeno* koordinato 1 kot element v četrti vrstici/dimenziji. To nam omogoča, da lahko v sklopu ene same transformacije vektorja oz. kvaterniona s to sestavljeno matriko izvedemo hkrati rotacijo in translacijo telesa v enem samem koraku.

Nad njimi izvajamo podobne operacije kot pri $3x3$ matrikah, le da imamo tu dodatno dimenzijo (dodatni stolpec in v ozadju tudi vrstico).

Pri matrikah bomo poleg osnovnih naštetih operacij potrebovali tudi operacijo za zamenjavo baze matrike. Če zamenjamo bazo matrike z neko transformacijsko matriko M_o, potem hočemo poiskati tako transformacijsko matriko M_t', ki bo nad to novo izračunano oz. transformirano matriko imela popolnoma identični učinek, kot ga ima neka poljubna transformacijska matrika M_t nad matriko z nespremenjeno bazo. To storimo na sledeč način:

$$M_t' = M_o M_t M_o^{-1} \tag{8.1}$$

To je uporabno predvsem, ko želimo prestopati med reprezentacijo matrike v lokalnem in svetovnem koordinatnem sistemu

Naj omenimo, da so podatkovni tipi elementov vektorjev in matrik števila v plavajoči vejici oz. racionalna števila.

8.2 Delci

Sedaj lahko implementiramo osnovne delce oz. točkovne mase. Razred bo vseboval štiri primerke prej implementiranega razreda vektorjev: linearno pozicijo, hitrost in pospešek delca ter seštevek sil, ki delujejo na delec. Poleg tega potrebujemo še konstanto za upor in pa maso, ki je zaradi že omenjene učinkovitosti shranjena kot inverzna masa. Za vse omenjene parametre imamo standardne *setter* in *getter* metode, izjema je le konstanta vsote sil; tej lahko s pomočjo metode prištejemo neko silo, predstavljeno v obliki 3D vektorja. To prištevanje izvedemo na začetku pred vsako izrisano sličico in to za vse sile,

ki delujejo nad izbranim delcem. Tako dobimo vsoto vseh delujočih sil nad delcem, prikladno shranjeno v eni spremenljivki.

Po akumulaciji vseh sil lahko na podlagi teh izračunamo posodobljeno stanje oz. pozicijo delca. V primeru, da je inverz mase delca manjši ali enak 0, se integracije sploh ne lotimo, saj gre v tem primeru za nepremični delec (ga smatramo kot delec z neskončno maso). Sicer, najprej izračunamo pospešek delca glede na vsoto sil; vektorju pospeška prištejemo vektor vsote sil, pomnožen s konstanto inverza mase. Nato posodobimo njegovo hitrost tako, da ji prištejemo vektor pospeška, pomnožen z dolžino trajanja prikaza zadnje izrisane sličice zaslona (torej, gre za časovno integracijo pospeška oz. hitrosti). Nad novo pridobljeno hitrostjo izvedemo še upor z množenjem le-te s konstanto upora, potencirano na dolžino trajanje zadnje izrisane sličice. S tem dosežemo, da nad delcem izvedemo pravilen upor, vezan oz. relativen na pretečeni čas. Sedaj imamo dejansko hitrost delca, s katero lahko manipuliramo njegovo pozicijo. To zopet storimo z integracijo, in sicer vektor hitrosti pomnožimo s trajanjem zadnje izrisane sličice in ta vektor nato prištejemo poziciji, s čimer lahko efektivno transliramo delec. Za konec še ponastavimo konstanto vsote sil, saj se kopičenje sil v tej konstanti odvije ponovno vsako iteracijo programa.

8.3 Sile nad delci

Konstantne sile, kot so npr. težnost Zemlje, zlahka predstavimo s konstantnim vektorjem; v tem primeru bi bil ta enak $\begin{bmatrix} 0 & 9.81 & 0 \end{bmatrix}^T$. Za bolj kompleksne sile, pri katerih se njihovo obnašanje spreminja v odvisnosti od časa in/ali drugih parametrov, pa moramo razviti bolj napreden model njihove predstavitve in manipulacije. Večina parametrov je podanih v plavajoči vejici, tako kot elementi matrik/vektorjev.

Ker imamo veliko tipov sil, ki so odvisni od različnih dejavnikov in se posledično različno obnašajo in različno vplivajo na delce, bomo za vsak izbrani tip implementirali svoj razred. Vsak tip sile bo moral implementirati metodo,

kjer bo nad izbrano instanco delca in lahko tudi na podlagi trajanja zadnje iteracije programa (podamo ju kot argumenta metodi), vsoti sil tega delca prištel vektor sile, proizveden s strani tega tipa sile v podanem času trajanja.

Zaradi prostorske učinkovitosti smo izvedli mapiranje primerkov sil in delcev, nad katerimi se sile izvajajo tako, da imamo namenski razred, ki vsebuje množico vseh mapiranj referenc sil in referenc delcev. Preko tega razreda lahko torej v zbirko dodajamo oz. beležimo delovanje sil nad delci, jih odstranjujemo (z odstranitvijo celotne zbirke se znebimo vseh povezav), hkrati pa s sprehodom po tej zbirki enovito pokličemo posodobitvene metode posameznih tipov sil, in tako apliciramo sile, proizvedene v dani iteraciji, nad vsemi delci, ki so deležni vpliva posameznih sil. Naj še omenimo, da se lahko ista referenca instance sile uporablja nad večimi referencami delcev, le za vsakega moramo podati nov vnos v zbirko povezav sila-delec.

Sledijo razlage implementiranih tipov sil nad delci.

8.3.1 Gravitacija

Predstavlja implementacijsko najbolj preprosto obliko sile. Potrebujemo le eno konstanto, ki predstavlja vektor pospeška. Na ta način lahko realiziramo tudi prej omenjeno gravitacijo zemlje.

Proizvedena sila je enaka zmnožku vektorja pospeška in pa mase delca (glej Enačbo (2.1)). Torej pri tej (pa tudi pri številnih drugih) sili ignoriramo čas trajanje zadnje iteracije programa, zaradi že obrazloženih razlogov (gravitacijo obravnavamo kot konstantno silo).

8.3.2 Upor

Za predstavitev tega tipe sile uporabimo dve racionalni konstanti, ki predstavljata koeficienta upora, ki zmanjšujeta hitrost delca. Ob kreiranju instance tega razreda z njima določimo, kakšen upor bo deloval nad delcem.

Nad obnašanjem upora pa ima vpliv tudi sam izbran delec, natančneje

njegova hitrost. Zato moramo najprej pridobiti vektor njegove hitrosti in izračunati njegovo dolžino. To vrednost nato pomnožimo s prvim koeficientom in jo prištejemo kvadratu iste vrednosti, pomnoženim z drugim koeficientom upora. Dobljen rezultat negiramo pomnožimo z normalizirano hitrostjo delca in tako dobimo iskano silo (glej Enačbo (3.1)).

Delci že imajo parameter, ki določa upor, v obliki konstantne vrednosti. Če želimo bolj kompleksen upor, le-tega nastavimo malo pod 1 (zaradi že omenjenih računskih problemov) in namesto njega uporabimo silo upora.

8.3.3 Vzmet med dvema delcema

Tu gre za vzmet, vpeto med dvema delcema, torej potrebujemo referenco instanc obeh delcev; ker pa referenco ene instance mapiramo s silo že preko namenskega razreda, potrebujemo v tem razredu le eno referenco. Poleg tega potrebujemo še parametre, specifične za vzmeti. Ta parametra sta konstanta vzmeti (določa njeno obnašanje oz. tip) in pa dolžina vzmeti v začetnem stanju.

Za izračun sile potrebujemo najprej vektor vzmeti, ki je enak razliki vektorjev pozicij obeh delcev, med katerima se nahaja vzmet (glej Enačbo (4.4)). Nato dolžini vzmeti v njenem začetnem stanju odštejemo dolžino dobljenega vektorja, absolutno vrednost rezultata pa pomnožimo s konstanto vzmeti. Normaliziran vektor vzmeti pomnožimo z negacijo novo pridobljene konstante in dobljeni produkt predstavlja iskano silo (glej Enačbo (4.3)).

8.3.4 Vzmet med delcem in fiksno točko

Zadeva je podobna sili vzmeti med dvema delcema, razlika je le v parametrih, saj ima sila namesto reference na instanco delca sedaj referenco na fiksno točko podano v vektorski obliki. Pri izračunu sile tako namesto pozicije enega delca uporabimo pozicijski parameter.

8.3.5 Elastika med dvema delcema

Od vzmeti med dvema delcema se v parametrih razlikuje le v dolžini vzmeti, ki tu predstavlja dolžino vzmeti v trenutku, ko prične ta proizvajati silo. Elastika proizvede silo le, ko se raztegne, ne pa tudi ko se skrči. Zato izračun sile, v primeru, da je dolžina vektorja vzmeti manjša ali enaka dolžini vzmeti v trenutku pričetka proizvajanje sile, prekinemo.

8.3.6 Elastika med delcem in fiksno točko

Potrebna je enaka sprememba sile elastike med dvema delcema, kot smo jo naredili pri sili vzmeti, ko smo ji namesto enega delca direktno podali pozicijo.

8.3.7 Plovnost

Ta sila posnema obnašanje sile plovnosti, ki se izvaja nad ploskvijo tekočine (predstavlja gladino tekočine), ki je vodoravna s tlemi oz. leži v ravnini XZ. Njeni parametri so naslednji: volumen delca, gostota tekočine, maksimalna globina potopitve delca, preden ta proizvede maksimalno plovno silo (ki omogoča vertikalno mirovanje delca oz. njegovo plovnost) in pa višina ploskve tekočine (določena glede na ničelno točko koordinate y).

Kar se tiče izračuna, je najprej potrebno ugotoviti, kako globoko je delec potopljen, če sploh je. To storimo tako, da preberemo njegovo y komponento pozicije in jo primerjamo z vsoto višine gladine tekočine in pa prej omenjeno maksimalno globino. Če je vrednost komponente višja od vsote omenjene komponente, ta sila ne deluje nad izbranim delcem, saj se nahaja izven tekočine in lahko končamo izračun. Če je vrednost manjša, je delce popolnoma potopljen in nad njim izvršimo silo v smeri y koordinate z vrednostjo zmnožka volumna delca in gostote tekočine. Sicer pa gre za delno potopljen delec, v tem primeru pa moramo upoštevati Enačbo (4.6).

8.4 Kontakti med delci

Kontakte med delci, torej delce, med katerimi pride do trka oz. medsebojnega prekrivanja, beležimo v svojem razredu. Zato potrebujemo referenco obeh instanc delcev oz. le enega v primeru, da imamo kontakt delca z okolico. Potrebujemo še koeficient povrnitve, globino penetracije kontakta, vektor smeri kontakta (gledano s strani prvega delca) in pa potrebna količino premika za posamezen delec, da delca nista več v preseku. Kontaktov ne akumuliramo, tako kot smo to storili pri silah.

V sklopu razreševanja kolizije moramo najprej poračunati njen sunek sile. Najprej potrebujemo ločevalno hitrost; v primeru, da je ta manjša od 0, to pomeni, da je kontakt stalen (pri mirujočih delcih) ali pa se kontakt že razrešuje in nadaljnja logika ni potrebna. V nasprotnem primeru moramo izračunati novo ločevalno hitrost, ki je enaka prej izračunani, pomnoženi z negiranim koeficientom povrnitve. Nato izračunamo količino hitrosti, ki se je nabrala izključno kot posledica pospeška (razlika obeh pospeškov), za primere mirujočih kontaktov. Če pri tem dobimo približevalno hitrost, moramo le-to odstraniti od ločevalne hitrosti (potrebno je paziti, da ne gremo v negativo in da upoštevamo koeficient povrnitve). Ker na spremembo v hitrosti delcev po kontaktu vpliva njuna masa, potrebujemo njun seštevek in v primeru, da je ta enak 0, gre za nepremična delca in lahko postopek prekinemo. Izračunamo potreben sunek sile za razrešitev kontakta (glej Enačbo (5.6)), ga pomnožimo z normalo kontakta (sunek sile na enoto inverza mase) in nato uporabimo v enačbi (glej Enačbo (5.7)), ki jo izvedemo za oba delca posebej (prvemu prištejemo, drugemu pa odštejemo hitrost).

Potrebujemo še mehanizem, s katerim ločimo delca v primeru, ko nahajata v preseku, kajti to predstavlja nerealno situacijo in moramo pravilno obnašanje pred izvedbo ločevalne hitrosti to penetracijo odpraviti (ju postaviti v stanje medsebojnega dotikanja). Če je ta manjša ali enaka od 0, penetracije ni in lahko zaključimo. Sicer jo pomnožimo s skupno maso delcev in normalo

kontakta, da dobimo količino razrešitve penetracije na enoto inverza mase. Sedaj lahko izračunamo potrebni količini premika za razrešitev prekrivanja tako, da dobljeno vrednost preprosto pomnožimo z inverzom mase posameznega delca (zopet je pri drugem delcu izraz negiran). Poziciji posameznega delca prištejemo ustrezno količino premika in delca nista več v preseku.

Kontakte upravljamo v posebnem razredu. V njem lahko določimo največje možno število iteracij odpravljanja kontaktov (največ koliko kontaktov bomo dovolili obravnavati). Vsebuje metodo, ki ji podamo zbirko vseh kontaktov, do katerih je prišlo v zadnji iteraciji programa in pa časovno dolžino trajanja te iteracije. Na podlagi teh dve parametrov izvedemo algoritem, predstavljen v podpoglavju Vrstni red razreševanja trkov.

S to kodo lahko realiziramo povezave med delci, ki temeljijo na kontaktih. Tako kot pri silah tudi tu ločimo primere, kjer imamo opravka z dvema delcema in primere, kjer imamo opravka z enim delcem in eno fiksno točko v svetovnih koordinatah.

V prvih primerih zopet potrebujemo referenco obeh instanc delcev, zanima nas pa tudi trenutna dolžina oddaljenosti med delcema. Vsak tip povezave delcev mora implementirati metodo, ki proizvede kontakt, ki ohranja delca znotraj omejitev njune povezave. Podamo ji referenco na prej način definiran kontakt (zgeneriramo samo en kontakt, kar povsem zadošča). Metoda vrača vrednost, ki nam pove, ali je do kontakta prišlo ali ne.

Sledi opis implementiranih tipov kontaktov.

8.4.1 Vrv

Tu potrebujemo poleg trenutne razdalje med delcema še maksimalno dovoljeno dolžino med njima in pa koeficient povrnitve. V primeru, da ni prišlo do raztega (trenutni razmik med delcema je manjši od maksimalne možne dolžine te vrvi), ni potrebno storiti nič, sicer pa dodamo oba delca posredovani referenci instance kontakta. Izračunamo še normalo kontakta (enaka smeri trka), mu predamo koeficient povrnitve in globino penetracije (razlika trenutne in

maksimalne razdalje med delcema).

8.4.2 Drogovi

Tu imamo namesto maksimalne kar stalno oddaljenost med delcema, koeficient povrnitve pa je vedno enak 0 (ker predstavljamo fiksno, trdno povezavo). Od vrvi pa se razlikuje v odvisnosti od trenutne dolžine med delcema; če je enaka prej določeni dolžini, končamo izvajanje metode, če je večja od te, pa se izvedejo iste operacije kot pri vrveh (le da govorimo tukaj o stalni namesto maksimalni oddaljenosti delcev), če je manjša, pa še negiramo normalo in globino penetracije (slednje le zato, da pridobimo pozitivno vrednost).

8.5 Manipulacija delcev

Tako kot smo imeli pri silah poseben razred, ki se shranjeval zbirko vseh mapiranj sil in delcev, smo enako storili za združitev vseh do sedaj realiziranih konstruktov. Vsebuje prej omenjeni razred z zbirko mapiranj sil in delcev, samo zbirko vseh delcev in pa vseh v trenutni iteraciji proizvedenih kontaktov med njimi ter maksimalno dovoljeno število kontaktov, instanco razreda za odpravljanje kontaktov (lahko uporabimo za obravnavo vseh kontaktov) in pa število maksimalno dovoljenih kontaktov, ki ga posredujemo ravnokar omenjeni instanci razreda.

Razred omogoča procesiranje celotne prisotne fizike nad delci. Najprej nad celotno zbirko mapiranj sila-delec sprožimo posodobitev oz. akumulacijo določenih sil nad ustreznimi delci. Nato nad delci izvedemo integracijo in s tem posodobimo njihov položaj in preostale parametre, kot posledica vsote sil nad posameznim delcem (klic te metode že poskrbi za ponastavitev vsote sil). Zatem zgeneriramo vse prisotne kontakte - sprehodimo se preko vseh kontaktov med delci (vrvi, drogovi) in kličemo njihovo metodo za generiranje kontakta, te kontakte pa shranjujemo v interno zbirko (spomnimo, da reference posameznih kontaktov posredujemo tej klicani metodi, ki ta kontakt nastavi). Na podlagi

vrnjene vrednosti lahko spremljamo kdaj/če presežemo maksimalno dovoljeno število kontaktov in v tem primeru prekinemo njihovo nadaljnje kopičenje. V primeru, da je prišlo do kontaktov, jih nato tudi razrešimo, še prej pa določimo število iteracij razreševanja kontaktov (kot smo omenili, smo se odločili za dvakratnik vseh trenutno prisotnih kontaktov).

8.6 Toga telesa

Poleg vsega kar vsebuje razred za delce, vsebuje ta razred še druge parametre, vezane na rotacije. Imamo 3D vektor kotne hitrosti, kvaternion orientacije telesa, konstanto kotnega upora, vektor vsote vseh navorov izvedenih nad telesom in pa inverz vztrajnostnega tenzorja (namesto inverza mase), podano tako v koordinatah telesa kot v koordinatah sveta (slednja se spreminja in je prisotna zaradi učinkovitosti - da se izognemo odvečnemu večkratnemu računanju le-te).

Vključili smo tudi transformacijsko matriko za pretvorbo iz lokalnega v svetovni koordinatni sistem, izključna zaradi boljše učinkovitosti - na ta način se izognemo ponovnemu izračunavanju le te vsakič, ko jo potrebujemo. Namreč, transformacijska matrika se znotraj posamezne iteracije programa ne spreminja.

Sama integracija pa je sledeča. Najprej poračunamo pospešek telesa kot produkt vsote vseh sil delujočih nad telesom in pa inverza mase, hkrati pa tudi kotni pospešek kot transformacija oz. množenje inverza vztrajnostnega tenzorja v koordinatah sveta z vektorjem vsote navorov, ki delujejo nad telesom v trenutni iteraciji.

Sedaj lahko posodobimo linearno hitrost telesa kot posledico spremembe linearnega pospeška in s tem sunkov sil. Ta je enaka množenju pospeška z dolžino trajanja prejšnje iteracije. Enak je izračun nove kotne hitrosti, le da namesto linearnega uporabimo kotni pospešek.

Nad obema hitrostma izvedemo še upor, ki je enak linearnemu uporu v

prvem oz. kotnem uporu v drugem primeru, potenciranim s časom trajanja zadnje izvršene iteracije programa.

Preostane nam še posodobitev pozicije in orientacije telesa tako, da pomnožimo dobljeno hitrost oz. dobljeno kotno hitrost s trajanjem zadnje iteracije programa.

Da pa vektor orientacije dejansko predstavlja orientacijo, ga moramo normalizirati, da si zagotovimo njegovo enotskost. Sedaj lahko posodobimo transformacijsko matriko telesa z Enačbo (5.7), ki ji dodamo stolpec za predstavitev pozicije telesa in pa inverz vztrajnostnega tenzorja v svetovnih koordinatah (pri njenem računanju izhajamo iz inverza vztrajnostnega tenzorja telesa, podanega v njegovih lokalnih koordinatah). Pri slednjem gre pravzaprav za transformacijo oz. zamenjavo baze matrike (glej Enačbo (8.1)).

Za konec le še ponastavilo vsoto sil in vsoto navorov posameznih togih teles.

Omenimo še, da razred vsebuje dva načina apliciranja posameznih sil: sile lahko izvedemo nad nekim togim telesom tako, da je točka apliciranja te sile v svetovnih ali lokalnih koordinatah telesa. V slednjem primeru moramo točko apliciranja sile pretvoriti v svetovne koordinate za vsako iteracijo aplikacije znova. To storimo tako, da s transformacijsko matriko transformiramo vektor fiksne točke, podan v lokalnih koordinatah.

8.7 Sile nad togimi telesi

Struktura razredov sil nad togimi telesi je zgrajena po enakem principu kot pri silah and delci. Torej vsaka vrsta sile mora implementirati metodo, ki sprejme referenco na instanco togega telesa in proizvede silo, imamo pa tudi razred, ki je zadolžen za hrambo in manipulacijo zbirke mapiranj sila-togo telo.

8.7.1 Gravitacija

Koda je povsem ista, kot pri isti sili nad delci, le da je tukaj aplicirana nad togim telesom.

8.7.2 Vzmeti

Koda je zelo podobna različici za vzmeti, razlikuje se le v izračunu vektorja vzmeti. Namreč, delci so predstavljeni z natanko eno točko, toga telesa pa se smatrajo kot telesa z nekim volumnom, in možnost izbire točke, nad katero bomo proizvedli neko silo, je ogromno (celotna površina telesa). Zato rabimo temu tipu sile dodati dve dodatni informaciji, in sicer sta to vektorja, ki označujeta neko točko na površini instance telesa, podano v lokalnih koordinatah istega telesa, ki predstavlja točko, nad katero deluje sila. V tem primeru je to vzmet; torej nam ti dve točki povesta, kje na površini obeh togih teles, ki sta povezani s silo vzmeti, se ju ta sila pripenja oz. na katerem delu telesa se aplicira. Ko ti vrednosti pretvorimo (s pomočjo transformacijske matrike) v svetovne koordinate, ju lahko nato odštejemo in dobimo želeni vektor vzmeti. Silo dodamo telesu skupaj z njeno aplikacijsko točko, navedeno v lokalnem koordinatnem sistemu telesa.

8.7.3 Plovnost

Tu potrebujemo poleg vseh parametrov v različici z delci še vektor, ki predstavlja središče plovnosti togega telesa v lokalnih koordinatah telesa. Implementacija je popolnoma enaka kot pri različici z delci, le da tu uporabimo y komponento vektorja središča plovnosti in ne vektorja pozicije telesa, hkrati pa uporabimo središče plovnosti kot fiksno točko, nad katero se izvede izračunana sila. Za čimbolj realno obnašanje teles na vodi se lahko poslužimo večih različnih instanc tega tipa sile, ki imajo središče aplikacije sile (v tem primeru središče plovnosti) pametno razporejene po celotnem telesu. Enako velja za druge tipe sil.

8.7.4 Aerodinamika

Za imitacijo aerodinamike potrebujemo vztrajnostni tenzor aerodinamike dimenzije $3x3$ in pa pozicijo njegovega apliciranja na togem telesu (lokalne koordinate). Z njim lahko izračunamo skupek sile, ki deluje na površino telesa kot posledica vetra. Uporabimo tudi referenco na vektor, ki predstavlja hitrost vetra (predstavlja parameter celotnega sveta).

Najprej moramo poračunati hitrost telesa, ki je enaka seštevku vektorjev hitrosti togega telesa in hitrosti vetra. Dobljeno hitrost preko inverza rotacijskega dela transformacijske matrike telesa s transformacijo pretvorimo v njegove lokalne koordinate (inverz zato, ker iz svetovnih pretvarjamo v lokalne koordiante telesa, samo rotacijski del matrike (dimenzija $3x$) pa uporabimo zato, ker vektor hitrosti predstavlja orientacijo (in količino hitrosti) in ne rotacijo in ima zato 3 komponente (ne gre za kvaternion)).

Ta korak je bil potreben zato, ker so tenzorji (kot smo že videli pri vztrajnostnem tenzorju) podani v lokalnih koordinatah in morajo biti zato v teh koordinatah tudi vsi v račun udeleženi vektorji oz. matrike. Sedaj lahko z aerodinamičnim tenzorjem transformiramo v lokalne koordinate pretvorjeni vektor izračunane hitrosti telesa, kar nam da silo podano v lokalnih koordinatah telesa.

Dobljen vektor sile zopet transformiramo z rotacijskim delom transformacijke matrike, a tokrat brez matričnega inverza, saj pretvarjamo predstavitev sile iz lokalnih v svetovne koordinate. Pretvorjeno silo skupaj z lokalno točko aplicirane sile dodamo izbranemu togemu telesu.

8.7.5 Kontrola aerodinamične sile

Za samo kontroliranje apliciranje aerodinamične sile, potrebujemo kontrolne površine, s katerimi lahko manipuliramo vpliv te sile na togo telo. Predstavlja nadgradnjo prej omenjenega razreda, ki poleg omenjenega tenzorja, ki tukaj predstavlja mirujoči položaj kontrolne površine, vsebuje še dodatna dva, ki

predstavljata tenzor površine v momentu minimalne oz. maksimalne vrednosti kontrole.

Za določanje oz. izbiranje količine uporabe posameznega tenzorja imamo še eno konstanto, s katero nastavljamo kontrolo površine, ki sprejme vrednosti na intervalu $[-1, 1]$, pri čemer -1 pomeni izključno uporabo minimalnega tenzorja, 1 izključno uporabo maksimalnega tenzorja, 0 izključno tenzor mirujoče kontrolne površine, sicer pa se uporabi interpolacija med dvema najbližjima tenzorjema glede na proporcionalno vrednost konstante kontrole površine. Zatem lahko izvedemo identične operacije, kot smo jih izvedli pri prejšnjem razredu (kličemo metodo nadrazreda, ki mu posredujemo na novo izračunani tenzor).

8.8 Detekcija trkov

Sam sistem za zaznavanje trkov med togimi telesi paroma preverja telesa, ali so v kontaktu in v primeru da sta, vrne informacijo o povzročenih kontaktih. Podatke o kontakti podajo generatorji kontaktov posameznih primitivnih teles, ki predstavljajo toga telesa kompleksnejših oblik. Gre za nadgradnjo že obravnavanih kontaktov.

Vse podatke, relevantne za kontakte med vsemi telesi, zberemo v svojem razredu. V njem imamo razširljivo zbirko, namenjeno hranjenju vseh proizvedenih kontaktov nad telesi.

8.8.1 Generator kontaktov

Zaradi učinkovitosti in preprostosti smo napisala nekaj razredov, ki omogočajo generiranje kontaktov za primitivne oblike teles. Vsem pa lahko pripišemo neke skupne parametre in pa operacije. Vsak primitivni tip telesa potrebuje referenco na togo telo, ki ga predstavlja v očeh detektorja trkov (njegovo pozicijo in orientacijo) in pa zamik primitivnega telesa od središča dejanskega togega telesa, predstavljen kot transformacijska matrika. Namreč, mi uporabljamo

kot središče togega telesa njegovo masno središče, ki pa se po navadi ne ujema z njegovim geometrijskim središčem in zato potrebujemo transformacijo primitivne geometrije glede na središče togega telesa. Posamezen primitiven tip primerjamo z nekim drugim, lahko istim tipom, in vse ustvarjene kontakte dodamo zbirki vseh kontaktov v namenskem razredu oz. strukturi. V primeru, da presežemo vnaprej določeno maksimalno dovoljeno število kontaktov v celotni aplikaciji, se nadaljnji generatorji kontaktov prenehajo izvajati, odvečni kontakti pa se tako ignorirajo.

8.8.1.1 Ploskev

Ploskev tu ni mišljena kot še eno primitivno telo, niti ne togo telo. Predstavlja nepremično telo, ki ga uporabimo pri kontaktih s primitivi (uporabno recimo za simulacijo tal ali pa zidu). Vsebuje le podatka o njenem zamiku od središka koordinatnega izhodišča sveta in pa vektor normale (njena usmerjenost).

8.8.1.2 Sfera

Za primitivni tip sfere potrebujemo poleg pozicije njenega središča še njen polmer.

Pri preverjanju kontaktov sfera-sfera preverimo, ali je razdalja med središčema sfer manjša ali enaka vsoti njunih polmerov. V primeru, da ni, ne proizvedemo kontaktov, sicer pa izračunamo njuno normalo kontakta, točko apliciranja kontakta, globino penetracije (vsoti obeh polmerov odštejemo oddaljenost središč sfer) ter iz teh podatkov, skupaj s koeficientom povrnitve, ustvarimo kontakt, ki ga dodamo med zbirko ostalih.

Pri sfera-ploskev preverjanju kontaktov ugotovimo ali je prišlo do penetracije tako, da izvedemo skalarni produkt pozicije sfere in normale ploskve, od katerega odštejemo polmer sfere ter zamik ploskve in če je ta vrednost enaka ali manjša 0, imamo dotik. Normala kontakta je kar enaka normali ploskve, penetracija pa negirani vrednosti (zato, da dobimo pozitivno vrednost) njune

prej izračunane razdalje. Točka apliciranja kontakta pa je enaka normali ploskve pomnoženi z vsoto globine penetracije in polmera sfere, ki jo odštejemo od pozicije sfere.

8.8.1.3 Kvader

Poleg pozicije njenega središča imamo tukaj še polovične dolžine vseh njenih stranic, shranjene v obliki vektorja (posamezna koordinata označuje dolžino stranic vzdolž posamezne osi). Polovične so zato, ker je središče primitivnih teles v njihovem geometričnem središču, in se zato te razdalje raztezajo do roba telesa, kar je bolj uporabno.

Pri kvader-ploskev preverjanju kontaktov, se sprehodimo skozi vsa oglišča kvadra, ki jih izračunamo z združitvijo transformacije togega telesa, zamika primitivnega kvadra in pa polovičnih dolžin njegovih stranic. Posamezno v svetovnih koordinatah podano točko nato pomnožimo z normalo ploskve, da dobimo oddaljenost med to točko in ploskvijo. Če je ta vrednost večja od zamika ploskve od izhodišča koordinatnega sistema, do kontakta pri tem oglišču ni prišlo in lahko zaključimo, sicer pa proizvedemo kontakt. Točka apliciranja kontakta je enaka normali ploskve, pomnoženi z vsoto zamika ploske ter prej izračunane razdalje med ploskvijo in točko, dobljeni vrednosti pa še prištejemo pozicijo točke. Normala kontakta je tudi tukaj, tako kot pri kontaktu med sfero in ploskvijo, enaka normali ploskve, globina penetracije pa je enostavno razlika med zamikom ploskve ter razdaljo med njo in trenutnim ogliščem.

Pri kvader-sfera preverjanju kontaktov bomo vedno imeli največ samo en kontakt - sfera se s svojo ploskvijo lahko dotika kvadra v neki njegovi točki, stranici ali ploskvi. Najprej poiščemo oglišče kvadra, ki je najbližji geometričnemu središču sfere. To storimo tako, da najprej preko transformacijske matrike kvadra (pridobimo jo na enak način kot v prejšnjem primeru) transformiramo center sfere v lokalne koordinate kvadra (transformiramo z inverzom transformacijske matrike, saj pretvarjamo iz svetovnega v lokalni koordinatni sistem). S tem si olajšamo računanje, saj se relativno na sfero znebimo orien-

tacije kvadra. Nato lahko s primerjavo posameznih koordinat središča sfere in pa posameznih dolžin zamikov oglišč vzdolž posamezne baze (torej polovične dolžine stranic, shranjene v vektorju) najdemo oglišče, najbližje središču sfere. Dobljeno oglišče pretvorimo nazaj v svetovni koordinatni sistem preko transformacijske matrike kvadra. Če je razdalja med središčem in to točko oz. ogliščem manjša ali enaka polmeru sfere, potem imamo kontakt in to oglišče predstavlja točko apliciranja kontakta, sam vektor med njima predstavlja normalo kontakta (ga prej še normaliziramo seveda), globina penetracije pa je enaka razliki med polmerom sfere in dolžino vektorja med omenjenima točkama.

Pri kvader-kvader preverjanju kontaktov, uporabimo algoritem imenovan *test ločevanja osi (SAT - separating axis test).* Z njim lahko zelo učinkovito ugotovimo ali sta dve telesi v kontaktu - če med njima poteka ločevalna os, potem ga ni. Gre pravzaprav za projekcijo 3D telesa na posamezne osi koordinatnega sistema. To storimo tako, da izberemo maksimalno in minimalno točko, ki se pojavi na telesu vzdolž posamezne osi. Če se ta intervala teles prekrivata za posamezne osi, potem obstaja možnost kontakta (če se ne prekrivata v nobeni osi, potem ga zagotovo ni, sicer pa ni nujno, da je dejansko kontakt prisoten). Zato potrebujemo izvršiti več testov in če katerikoli (vsaj en) uspe, potem se telesi ne dotikata, sicer pa lahko sklepamo, da gre za kontakt. Tu je zelo pomembno da pravilno izberemo osi za preverjanje. V splošnem moremo upoštevati naslednje osi - normala vsake izmed ploskev obeh teles, vektor, ki je pravokoten na posamezne pare stranic (par sestavlja ena stranica prvega in ena drugega telesa), za konec pa eliminiramo vse osi, ki so enake ali le negirane. Omenjena strategija deluje za vse konveksne mnogokotnike.

V trenutnem primeru, kjer imamo opravka z dvema kvadroma, bomo ločili primere, ko sta kvadra v kontaktu z robovi ali s stranicami, primer, ko se sekata rob enega in stranica drugega in primer, ko se oglišče enega seka s stranico drugega (kontakte oglišče-oglišče in oglišče-stranica lahko izpustimo, ker ne dodata novih ločevalnih osi). Nato se sprehodimo po vseh določenih ločevalnih oseh in ko naletimo na prvo, pri kateri uspe test ločevanja osi, lahko

algoritem zaključimo brez proizvoda kontakta. V nasprotnem primeru, če noben test ne uspe, imamo med telesoma kontakt.

Poglavje 9

Uporaba fizikalnega pogona v aplikaciji

Zmožnosti razvitega fizikalnega pogona bomo demonstrirali z aplikacijo, ki temelji na tem pogonu. Aplikacija vsebuje štiri različne scene, in vsaka izmed njih se osredotoča oz. demonstrira neko podmnožico izbranih fizikalnih področij, predstavljenih v tem diplomskem delu. V nadaljevanju bomo predstavili vsako izmed njih. Povedali bomo, kako izvajati interakcije v posamezni sceni oz. kako jo opravljati ter na kratko opisali njeno delovanje.

9.1 Upravljanje aplikacije

Ko poženemo aplikacijo, se začne izvajati *Scena 1.* Zamenjavo scene izvedemo z vrtenjem srednjega miškinega gumba, tako se s premikom naprej začne izvajati scena *Scena 2*, s premikom nazaj pa *Scena 4* (kot že rečeno imamo pet scen, in če gremo z vrtenjem miškinega gumba preko meje, se začne izvajati prva oz. zadnja scena). Aplikacijo zapremo s pritiskom na tipko *Esc*.

9.2 Scena 1

Tu prikažemo uporabo navadnih delcev iz poglavja Delci. Delci se izkoristijo za realicijo ognjemeta.

9.2.1 Upravljanje Scene 1

V prvi sceni imamo prvoosebno kamero (first person). Po sceni se lahko navigiramo/sprehajamo prosto v vse smeri, edina omejitev je, da ne moremo iti pod točko 0 v smeri osi y (višina). Premike izvedemo z naslednjimi tipkam:

- **'w':**

  Premik naprej.

- **'s':**

  Premik nazaj.

- **'a':**

  Premik bočno v levo stran.

- **'d'**

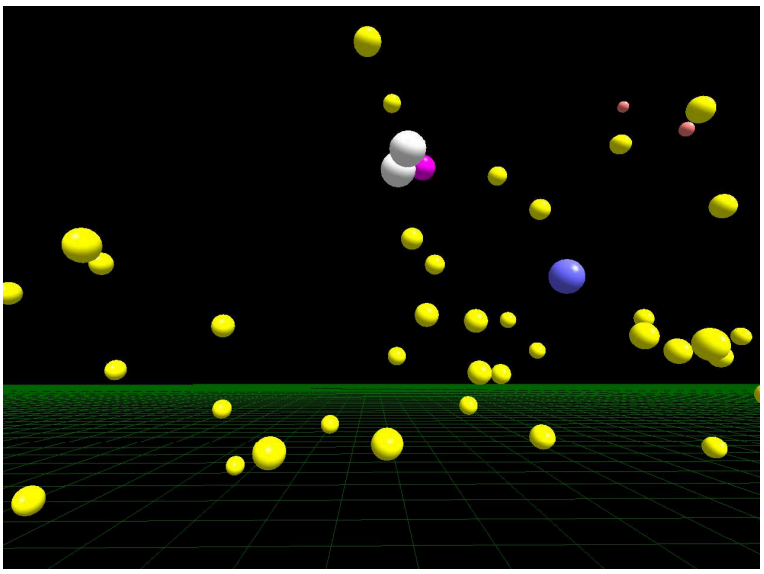
  Premik bočno v desno stran.

Smer pogleda oz. kamere pa usmerjamo z miško.

9.2.2 Opis delovanja Scene 1

Sama kamera se na začetku usmeri proti lokaciji, kjer bodo ob pritiskih različnih tipk na tipkovnici, nastali in se pognali v zrak delci različnih barv, z različno

začetno lokacijo, smerjo gibanja, pospeškom in časom trajanja oz. prisotnosti v sceni. Posamezne vrste izstrelkov ustvarimo in poženemo v tek s pritiskom na eno izmed številk tipkovnice od vključno 1 do 9, pri čemer vsaka predstavlja drugačen tip delca.



Slika 9.1: Posnetek ekrana Scene 1.

9.3 Scena 2

V tej sceni prikažemo delovanje masno celostnega dela fizikalnega pogona. Predstavlja viseči most in pa kroglo, ki leži na njem in jo lahko premikamo.

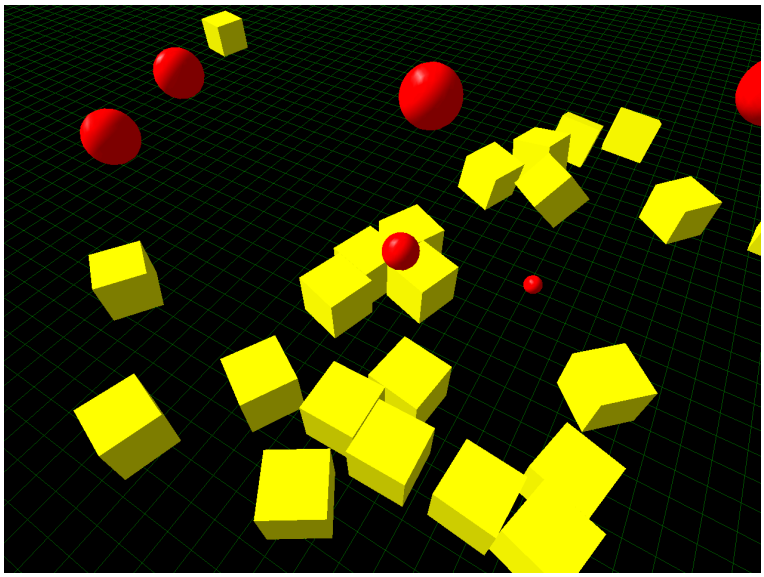
9.3.1 Upravljanje Scene 2

Samo kroglo lahko premikamo po celotni površini mostu. To storimo s smernimi kazalci na tipkovnici, pri čemer vsaka predstavlja premik v neko stran.

9.3.2 Opis delovanja Scene 2

Most je zgrajen iz masnih delcev in zanj veljajo trdnostne omejitve. Sestavljen je iz masnih delcev, ki so paroma povezana s t. i. drogovi (nedeformljivimi vezmi) - povezani so vzdolž mostu s sosednjim levim in desnim masnim delcem, če ta obstaja, in pa prav tako preko površine mostu z masnim delcem, ki se nahaja nasproti temu. Prav tako so na vsakega izmed njih pritrjeni kabli (deformljive vezi), ki imajo drugi konec statično lociran neposredno nad njimi in na ta način držijo celotno strukturo mostu nad tlemi. Kabli in drogovi torej predstavljajo trdnostne omejitve.

Ker ima krogla svojo neko maso (poleg upora, pozicije, hitrosti, pospeška (ga nima)), bo njeno premikanje vplivalo na premikanje oz. obnašanje samega mostu. To stori tako, da krogla proizvede kontakte nad najbližjim prej omenjenim masnim delcem mostu. Seveda, da krogla obstane na površini mostu, je potrebno proizvesti tudi kontakt med kroglo in površino.



Slika 9.2: Posnetek ekrana Scene 2.

9.4 Scena 3

Tu imamo opravka z masnimi delci, detekcijo trkov in razreševanjem le-teh med različnimi tipi oz. oblikami teles. Scena vsebuje 25 kock, ki jih lahko preko fizike transformiramo z uporabo okroglih izstrelkov.

9.4.1 Upravljanje Scene 3

Enako kot v Sceni 1, imamo tudi tu prvoosebno kamero s povsem enakimi kontrolami tipkovnice in miške. Poleg tega pa lahko z mesta, kjer se trenutno nahajamo v 3D prostoru in v smer, kamor gledamo (oz. z mesta, kjer se nahaja kamera in smeri, kamor je ta usmerjena) izstrelimo oz. pošljemo v sceno krogle s pritiskom na levo ali desno tipko miške. Tip izstrelka izberemo s pritiskom na eno izmed številk od 1 do 5.

9.4.2 Opis delovanja Scene 3

Izstrelki oz. krogle imajo različne lastnosti. Vsak izstrelek je določen z naslednjimi lastnostmi - velikost (volumen), masa, pospešek, upor, vztrajnostni moment in začetna pozicija ter usmerjenost, ki sta določeni s pozicijo in usmerjenostjo kamere v trenutku izstrelitve.

Če izstrelki trčijo med sabo, se njihovi trki razrešijo, prav tako velja za trke med kockami ter trke med njimi in izstrelki. Enako velja za trke med kockami in tlemi.

9.5 Scena 4

V zadnji simulaciji želimo demonstrirati uporabo togih teles. To storimo s simulacijo letala.

9.5.1 Upravljanje Scene 4

Simulacija se začne z letalom, ki se požene v tek. Pilotiramo z naslednjimi tipkami:

- **'w':**

  Nagib navzdol.

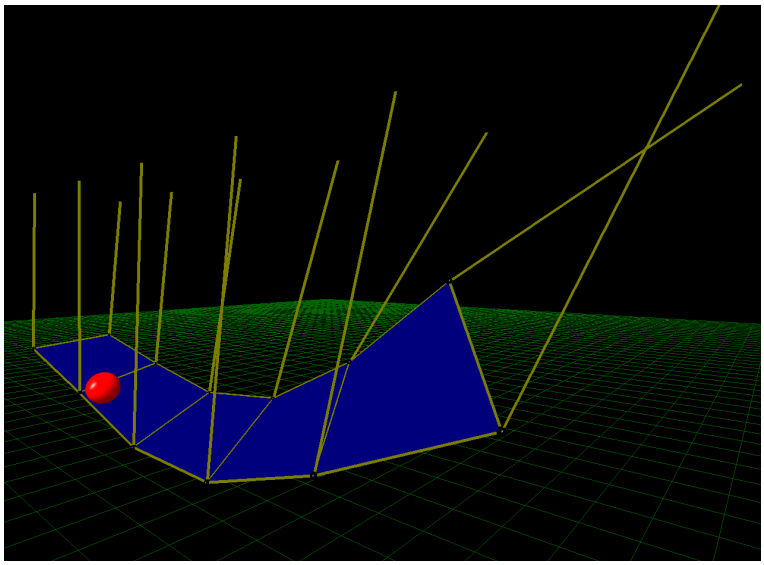
- **'s':**

  Nagib navzgor.

- **'a':**

  Zavoj levo s krili.

- **'d'**



Slika 9.3: Posnetek ekrana Scene 3.

Zavoj desno s krili.

- **'q':**

  Zavoj levo z repnim krmilom.

- **'e'**

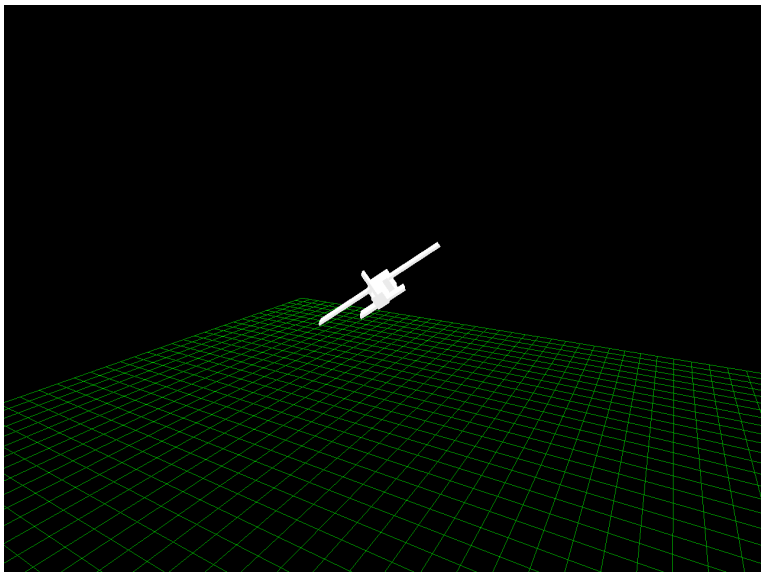
  Zavoj desno z repnim krmilom.

- **'r'**

  Ponastavitev simulacije na začetno stanje.

9.5.2 Opis delovanja Scene 4

Nad letečim letalom delujejo sila gravitacije in pa aerodinamične sile. Samo letalo ima štiri kontrolne površine, in sicer dve krili, repno krmilo in repno krilo. Vse razen repnega krmila lahko upravljamo preko zgoraj omenjenih kontrol (se pa uporabi za generiranje sil). S krili obračamo letalo okoli osi z (roll) ali okoli osi x (pitch), z navpičnim repnim krmilom pa okoli osi y (jaw).



Slika 9.4: Posnetek ekrana Scene 4.

Slike

Literatura

[1] Adrian Boeing and Thomas Bräunl. Evaluation of real-time physics simulation systems. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, GRAPHITE '07, pages 281–288, New York, NY, USA, 2007. ACM.

[2] Jeffrey Chin, Richard Dukes, and William Gamson. Assessment in simulation and gaming: A review of the last 40 years. *Simulation & Gaming*, 40(4):553–568, 2009.

[3] WulfG. Dettmer and Djordje Peric. On the coupling between fluid flow and mesh motion in the modelling of fluid-structure interaction. *Computational Mechanics*, 43(1):81–90, 2008.

[4] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[5] M. Hain and P. Wriggers. Numerical homogenization of hardened cement paste. *Computational Mechanics*, 42(2):197–212, 2008.

[6] Ladislav Kavan, Dan Gerszewski, Adam W. Bargteil, and Peter-Pike Sloan. Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph.*, 30(4):93:1–93:10, July 2011.

[7] Ladislav Kavan, Dan Gerszewski, Adam W. Bargteil, and Peter-Pike Sloan. Physics-inspired upsampling for cloth simulation in games. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 93:1–93:10, New York, NY, USA, 2011. ACM.

[8] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. Eurographics, 2001.

[9] Florian Lettner. Physics in games.

[10] WingKam Liu and hal McVeigh. Predictive multiscale theory forsign of heterogeneous materials. *Computational Mechanics*, 42(2):147–170, 2008.

[11] Murat Manguoglu, Kenji Takizawa, AhmedH. Sameh, and TayfunE. Tezduyar. Solution of linear systems in arterial fluid mechanics computations with boundary layer mesh refinement. *Computational Mechanics*, 46(1):83–89, 2010.

[12] Ian Millington. *Game physics engine development*. Morgan Kaufmann Publishers Amsterdam, 2007.

[13] Sunil Sathe and TayfunE. Tezduyar. Modeling of fluid-structure interactions with the space-time finite elements: contact problems. *Computational Mechanics*, 43(1):51–60, 2008.

[14] Roger Smith. The long history of gaming in military training. *Simulation & Gaming*, 41(1):6–19, 2010.

[15] Kenji Takizawa, Bradley Henicke, TayfunE. Tezduyar, Ming-Chen Hsu, and Yuri Bazilevs. Stabilized space-time computation of wind-turbine rotor aerodynamics. *Computational Mechanics*, 48(3):333–344, 2011.

[16] Tetsuya Uchiki, Toshiaki Ohashi, and Mario Tokoro. Collision detection in motion simulation. *Computers & Graphics*, 7(3-4):285–293, 1983.

Viri slik

[17] Ohranitev gibalne količine pri trkih. Leta 2014 dostopno na:
http://www.physicsclassroom.com/Class/momentum/u4l2e5.gif

[18] Razteg in skrčenje vzmeti. Leta 2014 dostopno na:
http://upload.wikimedia.org/wikipedia/commons/f/f0/
HookesLawForSpring-English.png

[19] Trk teles v 2D prostoru. Leta 2014 dostopno na:
http://www.silverlightshow.net/Storage/Users/MisterGoodcat/__
___image_2.png

[20] Sile delujoče na telo na klančini. Leta 2014 dostopno na:
http://i.stack.imgur.com/6Y5fC.gif

[21] Trigonometrične funkcije enotskega kroga. Leta 2014 dostopno na:
http://i.stack.imgur.com/s3ybv.png

[22] Sinusna funkcija. Leta 2014 dostopno na:
http://webgraphing.com/images/concept/transconc1.jpg

[23] Pravilo desne roke pri navoru. Leta 2014 dostopno na:
http://electron9.phys.utk.edu/Collisions/images/Image4.gif

[24] Delovanje navora. Leta 2014 dostopno na:
http://upload.wikimedia.org/wikipedia/commons/2/2e/Torque%2C_
position%2C_and_force.svg

[25] Navor. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Navor

[26] Navor. Leta 2014 dostopno na:
http://en.wikipedia.org/wiki/Torque

[27] Rotacijska os togega telesa. Leta 2014 dostopno na:
http://upload.wikimedia.org/wikipedia/commons/b/b4/Moment_of_inertia_rod_center.svg

[28] Fizika žiroskopa. Leta 2014 dostopno na:
http://www.real-world-physics-problems.com/images/gyroscope_physics_1.png

[29] Kardanska zapora. Leta 2014 dostopno na:
http://ncc.phinf.naver.net/ncc01/2011/3/24/61/4.jpg

[30] Masni delci telesa. Leta 2014 dostopno na:
http://www.ihatephysics.com/images/quantum_images/slit_mystery/happy_ball.gif

[31] Sila in posldični navor. Leta 2014 dostopno na:
http://www.4physics.com/phy_demo/newton/vtord.gif

[32] Navor pri viličastem ključu ob različnem prijemu. Leta 2014 dostopno na:
http://www.marylandmetrics.com/tech/torqueB.gif

[33] Plovnost telesa. Leta 2014 dostopno na:
http://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Buoyancy.svg/301px-Buoyancy.svg.png

[34] Rotacijske osi v 3D prostoru. Leta 2014 dostopno na:
http://upload.wikimedia.org/wikipedia/commons/7/7e/Rollpitchyawplain.png

[35] Enotska sfera. Leta 2014 dostopno na:
`http://www.gg.caltech.edu/STC/jpegs/figornew.jpg`

[36] Rotacijska os telesa. Leta 2014 dostopno na:
`http://upload.wikimedia.org/wikipedia/commons/b/b4/Moment_of_inertia_rod_center.svg`

[37] Sila in navor pri rotiranju matice z viličastem ključem. Leta 2014 dostopno na:
`http://www.marylandmetrics.com/tech/torqueA.gif`

Ostali viri

[38] GNU General Public Licence. Leta 2014 dostopno na:
https://www.gnu.org/copyleft/gpl.html

[39] Gibalna količina. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Gibalna_koli%C4%8Dina

[40] Ohranitev gibalne količine pri trkih. Leta 2014 dostopno na:
http://www.nauk.si/materials/4392/out/#state=2

[41] Gibalna količina. Leta 2014 dostopno na:
http://www.physicsclassroom.com/Class/momentum

[42] Tipi fizikalnih pogonov. Leta 2014 dostopno na:
http://cyborgdino.com/2012/04/the-mystery-behind-physics-engines/

[43] Kotna hitrost. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Kotna_hitrost

[44] Kvarternioni. Leta 2014 dostopno na:
http://zipcon.net/~swhite/docs/math/quaternions/
representations.html

[45] Vztrajnost. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Vztrajnost

[46] Vztrajnost. Leta 2014 dostopno na:
http://www.physicsclassroom.com/class/momentum/u4l1a.cfm

[47] Tenzor. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Tenzor

[48] Kotna vztrajnost togih teles. Leta 2014 dostopno na:
http://kwon3d.com/theory/moi/iten.html

[49] Vektorski produkt. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Vektorski_produkt

[50] Vztrajnostni moment. Leta 2014 dostopno na:
http://hyperphysics.phy-astr.gsu.edu/hbase/mi.html

[51] Vztrajnostni moment. Leta 2014 dostopno na:
http://en.wikipedia.org/wiki/Moment_of_inertia

[52] Vztrajnostni moment. Leta 2014 dostopno na:
http://sl.wikipedia.org/wiki/Vztrajnostni_moment

[53] Vztrajnostni tenzor. Leta 2014 dostopno na:
http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec26.pdf

[54] Vztrajnostni tenzor. Leta 2014 dostopno na:
http://www.math24.net/physical-applications-of-triple-integrals.html

[55] Vztrajnostni tenzor. Leta 2014 dostopno na:
http://farside.ph.utexas.edu/teaching/336k/Newtonhtml/node64.html

[56] Vztrajnostni tenzor. Leta 2014 dostopno na:
http://www.ro.feri.uni-mb.si/predmeti/robotika_2/predavanja/vztrajnostni_moment.pdf

[57] Vztrajnostni tenzor. Leta 2014 dostopno na:
`http://techhouse.brown.edu/~dmorris/projects/tutorials/inertia.tensor.summary.pdf`

[58] Eulerjevi koti. Leta 2014 dostopno na:
`http://sl.wikipedia.org/wiki/Eulerjevi_koti`

[59] Eulerjevi koti. Leta 2014 dostopno na:
`http://www.chrobotics.com/library/understanding-euler-angles`

[60] Kotna hitrost. Leta 2014 dostopno na:
`http://sl.wikipedia.org/wiki/Kotna_hitrost`

[61] Kotni pospešek. Leta 2014 dostopno na:
`http://sl.wikipedia.org/wiki/Kotni_pospesek`

[62] Kardanska zapora. Leta 2014 dostopno na:
`http://en.wikipedia.org/wiki/Gimbal_lock`